Deriving SN from PSN: a general proof technique

Emmanuel Polonowski

April 2006

TR–LACL–2006–5

Laboratoire d'Algorithmique, Complexité et Logique (LACL)
Département d'Informatique
Université Paris 12 – Val de Marne, Faculté des Science et Technologie
61, Avenue du Général de Gaulle, 94010 Créteil cedex, France
Tel.: (33)(1) 45 17 16 47, Fax: (33)(1) 45 17 66 01

Laboratory of Algorithmics, Complexity and Logic (LACL)
University Paris 12 (Paris Est)

Technical Report **TR–LACL–2006–5**

E. Polonowski.

Deriving SN from PSN: a general proof technique

Deriving SN from PSN: a general proof technique

Emmanuel Polonowski

Abstract

In the framework of explicit substitutions there is two termination properties: preservation of strong normalization (PSN), and strong normalization (SN). Since there are not easily proved, only one of them is usually established (and sometimes none). We propose here a connection between them which helps to get SN when one already has PSN. For this purpose, we formalize a general proof technique of SN which consists in expanding substitutions into "pure" λ-terms and to inherit SN of the whole calculus by SN of the "pure" calculus and by PSN. We apply it successfully to a large set of calculi with explicit substitutions, allowing us to establish SN, or, at least, to trace back the failure of SN to that of PSN.

Contents

1 Introduction

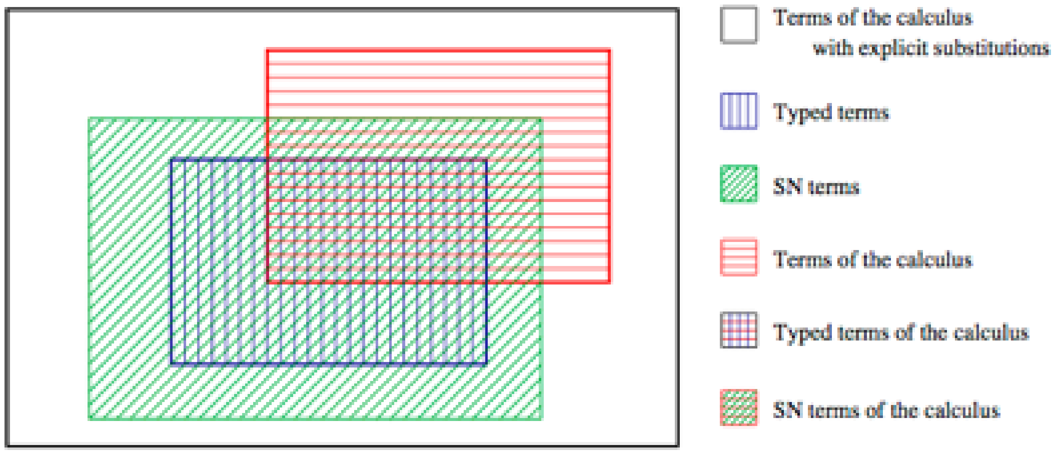
Calculi with explicit substitutions were introduced [1] as a bridge between λ-calculus [7, 2] and concrete implementations of functional programming languages. Those calculi intend to refine the evaluation process by proposing reduction rules to deal with the substitution mechanism – a *meta*-operation in the traditional λ-calculus. It appears that, with those new rules, it was much harder (and sometimes impossible) to get termination properties.

The two main termination properties of calculi with explicit substitutions are:

- **Preservation of strong normalization** (PSN), which says that if a pure term (*i.e.* without explicit substitutions) is strongly normalizing (*i.e.* cannot be infinitely reduced) in the pure calculus (*i.e.* the calculus without explicit substitutions), then this term is also strongly normalizing with respect to the calculus with explicit substitutions.

- **Strong normalization** (SN), which says that, with respect to a typing system, every typed term is strongly normalizing in the calculus with explicit substitutions, *i.e.* every terms in the subset of typed terms cannot be infinitely reduced.

These two properties are not redundant, and Fig. 1 shows the differences between them. PSN says that the horizontally and diagonally hatched rectangle is included in the diagonally hatched rectangle. SN says that the vertically hatched rectangle is included in the diagonally hatched rectangle. Even if they work on a different set of terms, there is a common part: the vertically and horizontally hatched rectangle, which represent the typed pure terms.



Terms of the calculus with explicit substitutions
Typed terms
SN terms
Terms of the calculus
Typed terms of the calculus
SN terms of the calculus

Figure 1: Termination properties

SN and PSN are both termination properties, although their proofs are not always clearly related: sometimes SN is shown independently of PSN (directly, by simulation, *etc.*, see for example [8, 10]), sometimes SN proofs uses PSN (see for example [4]). We present here a general proof technique of SN via PSN, initially suggested by H. Herbelin, which uses that common part of typed pure terms.

More formally, we may introduce the following notations: we denote Λ the set of λ-terms, Λ_T the set of typed Λ-terms with a given typing system T, Λ_{SN} the set of terminating Λ-terms (*i.e.* with a finite derivation tree); we denote Λ^X, Λ^X_T, Λ^X_{SN} the corresponding set for calculi with eXplicit substitutions.

By definition, we have the following set inclusions:

$$\begin{array}{ccc} \Lambda_T \subset \Lambda & \text{and} & \Lambda_{SN} \subset \Lambda \\ \Lambda^X_T \subset \Lambda^X & \text{and} & \Lambda^X_{SN} \subset \Lambda^X \\ \Lambda \subset \Lambda^X & \text{and} & \Lambda_T \subset \Lambda^X_T \end{array}$$

The usual strong normalisation property of typed λ-calculus gives

$$\Lambda_T \subset \Lambda_{SN}$$

As regard to calculi with explicit substitutions, we have the following properties. At first, the property PSN gives

$$\Lambda_{SN} \subset \Lambda_{SN}^{X}$$

At last, the strong normalization property of typed Λ^X-terms completes with the following inclusion:

$$\Lambda_{T}^{X} \subset \Lambda_{SN}^{X}$$

In the following section, we formalize a proof technique that exploits this diagram and in the remaining sections we apply this technique to a set of calculi. This set has been chosen for the variety of their definitions: with or without De Bruijn indices, unary or multiple substitutions, with or without composition of substitutions, and even a symmetric non-deterministic calculus. In the last section, we briefly talk about perspectives in this framework.

2 Proof Technique

The idea of this technique is the following. Let t be a typed term with explicit substitutions for which we want to show termination. With the help of its typing judgment, we build a typed pure term t' which can be reduced to t. For that purpose, we expand the substitutions of t into redexes. We call this expansion *Ateb* (the opposite of *Beta* which is usually the name of the rule which creates explicit substitutions). Then, with SN of the pure calculus and PSN, we can export the strong normalization of t' (in the pure calculus) to t (in the calculus with explicit substitutions).

In practice, this sketch will only apply in some cases, and some others will require some adjustment to this technique. For our technique to work, we need that the *Ateb* expansion satisfies some properties. The first one is always easily checked.

Property 2.1 (Preservation of typability) If t is typable, with respect to a typing system T, in the calculus with explicit substitution, then $Ateb(t)$ is typable, with respect to a typing system T' (possibly $T' = T$) in the pure calculus.

Only some calculi can exhibit an *Ateb* function which satisfies the second one.

Property 2.2 (Initialization) $Ateb(t)$ reduces to t in zero or more steps in the calculus with explicit substitutions.

If we can get it, then we use the direct proof to be presented in section 2.1. Otherwise, we need to use the simulation proof to be presented in section 2.2.

2.1 Direct proof

We can immediately establish the theorem.

Theorem 2.3 For all typing systems T and T' such that, in the pure calculus, all typable terms with respect to T are strongly normalizing, if there exists a function *Ateb* from explicit substitution terms to pure terms satisfying properties 2.1 and 2.2 then PSN implies SN (with respect to T').

Proof: For every typed term t of the calculus with explicit substitution, $Ateb(t)$ is a pure typed term (by property 2.1). By the strong normalization hypothesis of the typed pure calculus, we have $Ateb(t) \in \Lambda_{SN}$. By hypothesis of PSN we obtain that $Ateb(t)$ is in Λ^X_{SN}. By property 2.2, we get $Ateb(t) \to^* t$, which gives us directly $t \in \Lambda^X_{SN}$. ■

2.2 Simulation proof

We must relax some constraints on *Ateb*. We will try to find an expansion of t to t' such that t' reduces to a term u and there exists a relation $\mathcal{R}$ with $u\mathcal{R}t$. The chosen relation must, in addition, enable a simulation of the reductions of t by the reduction of u. If it is possible, we can infer strong normalization of t from strong normalization of u.

To proceed with the simulation, we first split the reduction rules of the calculus with explicit substitutions into two disjoints sets. The set R_1 contains rules which are trivially terminating, and R_2 contains the others. Secondly, we build a relation $\mathcal{R}$ which satisfies the following properties.

Property 2.4 (Initialisation) For every typed term t, there exists a term $u\mathcal{R}t$ such that $Ateb(t)$ reduces in 0 or more steps to u in the calculus with explicit substitutions.

Property 2.5 (Simulation *) For every term t, if $t \to_{R_1} t'$ then, for every $u\mathcal{R}t$, there exists u' such that $u \to^* u'$ and $u'\mathcal{R}t'$.

Property 2.6 (Simulation $^+$) For every term t, if $t \to_{R_2} t'$ then, for every $u\mathcal{R}t$, there exists u' such that $u \to^+ u'$ and $u'\mathcal{R}t'$.

We display those properties as diagrams :

Initialisation

$$\begin{tikzcd} & t \arrow[dl] \arrow[d, phantom, "\mathcal{R}"] \\ Ateb(t) \arrow[r, "*"] & u \end{tikzcd}$$

Simulation *

$$\begin{tikzcd} t \arrow[r, "R_1"] \arrow[d, phantom, "\mathcal{R}"] & t' \arrow[d, phantom, "\mathcal{R}"] \\ u \arrow[r, "*"] & u' \end{tikzcd}$$

Simulation $^+$

$$\begin{tikzcd} t \arrow[r, "R_2"] \arrow[d, phantom, "\mathcal{R}"] & t' \arrow[d, phantom, "\mathcal{R}"] \\ u \arrow[r, "+"] & u' \end{tikzcd}$$

With this material, we can establish the theorem.

Theorem 2.7 For all typing systems T and T' such that, in the pure calculus, all typable terms with respect to T are strongly normalizing, if there exists a function $Ateb$ from explicit substitution terms to pure terms and a relation $\mathcal{R}$ on explicit substitutions terms satisfying properties 2.1, 2.4, 2.5 and 2.6 then PSN implies SN (with respect to T').

Proof: We prove it by contradiction. Let t be a typed term with explicit substitutions which can be infinitely reduced. By property 2.4 there exists a term u such that $Ateb(t) \to^* u$, and $Ateb(t)$ is a pure typed term (by property 2.1). By the strong normalization hypothesis of the typed pure calculus, we have $Ateb(t) \in \Lambda_{SN}$. By hypothesis of PSN we obtain that $Ateb(t)$ is in Λ^X_{SN} and it follows that $u \in \Lambda^X_{SN}$.

By property 2.4, we also have $u\mathcal{R}t$, and, with properties 2.5 and 2.6, we can build an infinite reduction from u, contradicting the strong normalization of u. ∎

3 λx-calculus

The λx-calculus [6, 5] is probably the simplest calculus with explicit substitutions. It only makes the substitution explicit. Since this calculus provides no rules to deal with substitutions composition, it preserves strong normalization. It is for this calculus that the technique has been originate used by Herbelin. Therefore, we can without surprises apply the direct proof to get strong normalization.

3.1 Definition

Terms of the λx-calculus are given by the following grammar:

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid t[t/x]$$

Here follows the reduction rules:

$$\begin{array}{rcl}
(\lambda x.t)u & \to_{Beta} & t[u/x] \\
(t\ u)[v/x] & \to_{App} & (t[v/x])\ (u[v/x]) \\
(\lambda x.t)[u/y] & \to_{Lambda} & \lambda x.(t[u/y]) \\
x[t/x] & \to_{Var1} & t \\
y[t/x] & \to_{Var2} & y
\end{array}$$

The rule *Lambda* is applied modulo α-conversion of the bound variable x.

Here follows the typing rules:

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[u/x] : A}$$

$$\frac{\Gamma \vdash t : B \to A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A} \qquad \frac{\Gamma, x : B \vdash t : A}{\Gamma \vdash \lambda x.t : B \to A}$$

3.2 Strong normalisation proof

We define the *Ateb* function as follows:

$$\begin{array}{lcl}
Ateb(x) & = & x \\
Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\
Ateb(\lambda x.t) & = & \lambda x.Ateb(t) \\
Ateb(t[u/x]) & = & (\lambda x.Ateb(t))\ Ateb(u)
\end{array}$$

Remark that *Ateb* performs the exact reverse rewriting of the rule *Beta*. It straightforwardly follows that if $t' = Ateb(t)$ then $t' \to^*_{Beta} t$ and $Ateb(t)$ does not contain any substitutions.

We check that the $Ateb(t)$ is typable.

Lemma 3.1

$$\Gamma \vdash t : A \ \Rightarrow\ \Gamma \vdash Ateb(t) : A$$

Proof: By induction on the typing derivation of t. The only non-trivial case is that of substitutions. We have $t = u[v/x]$ and

$$\frac{\Gamma \vdash v : B \quad \Gamma, x : B \vdash u : A}{\Gamma \vdash u[v/x] : A}$$

By induction hypothesis, we have $\Gamma, x : B \vdash Ateb(u) : A$ and $\Gamma \vdash Ateb(v) : B$. We can build the typing derivation of $Ateb(t) = \lambda x.Ateb(u))\ Ateb(v)$ as follows

$$\frac{\dfrac{\Gamma, x : B \vdash Ateb(u) : A}{\Gamma \vdash \lambda x.Ateb(u) : B \to A} \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash (\lambda x.Ateb(u))\ Ateb(v) : A}$$

■

We can apply Theorem 2.3.

Corollary 3.2 Since the λx-calculus enjoys PSN [6] and the λ-calculus enjoys SN of simply-typed terms [11], we conclude that the λx-calculus enjoys SN of simply-typed terms.

4 $\lambda\upsilon$-calculus

The $\lambda\upsilon$-calculus [12, 3] is the De Bruijn counterpart of $\lambda\mathtt{x}$. As $\lambda\mathtt{x}$, it has no composition rules, and therefore satisfies PSN. For this calculus, we must use the simulation proof to deal with indices modification operators. We succeed to use it and it is, as far as we know, the first proof of SN for a simply typed version of $\lambda\upsilon$ (see [13]).

4.1 Definition

Terms of $\lambda\upsilon$-calculus are given by the following grammar:

$$\begin{array}{l} t ::= n \mid (t\ t) \mid \lambda t \mid t[s] \\ s ::= a/ \mid \Uparrow(s) \mid \uparrow \end{array}$$

Remark that a substitution is always build from a (possibly empty) list of $\Uparrow$ followed by either a $t/$, or a $\uparrow$. We will then write substitutions in a more general form: either $t[\Uparrow^i (t/)]$, or $t[\Uparrow^i (\uparrow)]$, where $\Uparrow^i (s)$ denotes $\underbrace{\Uparrow (\Uparrow (...(\Uparrow}_{i}(s))...))$.

Here follows the reduction rules:

$$\begin{array}{lll} (\lambda t)u & \to_{B} & t[u/] \\ (t\ u)[s] & \to_{App} & (t[s])\ (u[s]) \\ (\lambda t)[s] & \to_{Lambda} & \lambda(t[\Uparrow (s)]) \\ 1[t/] & \to_{FVar} & t \\ n+1[t/] & \to_{RVar} & n \\ 1[\Uparrow (s)] & \to_{FVarLift} & 1 \\ n+1[\Uparrow (s)] & \to_{RVarLift} & n[s][\uparrow] \\ n[\uparrow] & \to_{VarShift} & n+1 \end{array}$$

Here follows the typing rules (where $n = |\Gamma| + 1$) :

$$\frac{}{\Gamma, A, \Delta \vdash n : A} \qquad \frac{\Gamma \vdash s \rhd \Gamma' \quad \Gamma' \vdash t : A}{\Gamma \vdash t[s] : A}$$

$$\frac{\Gamma \vdash t : B \to A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A} \qquad \frac{B, \Gamma \vdash t : A}{\Gamma \vdash \lambda t : B \to A}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t/ \rhd A, \Gamma} \qquad \frac{}{A, \Gamma \vdash \uparrow \rhd \Gamma} \qquad \frac{\Gamma \vdash s \rhd B, \Gamma}{A, \Gamma \vdash \Uparrow (s) \rhd A, B, \Gamma}$$

4.2 Strong normalisation proof

We define the *Ateb* function as follows:

$$\begin{array}{lcl}
Ateb(n) & = & n \\
Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\
Ateb(\lambda t) & = & \lambda Ateb(t) \\
Ateb(t[u/]) & = & (\lambda Ateb(t))\ Ateb(u) \\
Ateb(t[\Uparrow^i (u/)]) & = & (\lambda I_i(Ateb(t)))\ K_i(Ateb(u)) \\
Ateb(t[\Uparrow^i (\uparrow)]) & = & J_i(Ateb(t))
\end{array}$$

Example 4.1 For instance, if we suppose that for any tt among t, u, v, w we have $tt = Ateb(tt)$, then we get

$$Ateb((t[u/]\ v[\Uparrow (\Uparrow (\Uparrow (w/)))])[\Uparrow (\Uparrow (\uparrow))]) \quad = \quad J_2(((\lambda t)u)\ ((\lambda I_3(v))K_3(w)))$$

Where $I_i(t))$, $K_i(t)$ and $J_i(t)$ are functions that we will define in the sequel. The intuition about those function is the following: substitutions perform some re-indexing of terms upon which they are applied, those functions intend to anticipate this re-indexing. To understand the necessity of those functions, let us look at some typing derivation. To begin with, we take $t[\Uparrow^i (u/)]$, where $\Delta = D_i, ..., D_1$ $(i = |\Delta|)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash u : B}{\Gamma \vdash u/ \triangleright B, \Gamma}}{D_1, \Gamma \vdash \Uparrow (u/) \triangleright D_1, B, \Gamma}}{\vdots}}{D_{i-1}, ..., D_1, \Gamma \vdash \Uparrow^{i-1} (u/) \triangleright D_{i-1}, ..., D_1, B, \Gamma}}{D_i, D_{i-1}, ..., D_1, \Gamma \vdash \Uparrow^i (u/) \triangleright D_i, D_{i-1}, ..., D_1, B, \Gamma} \quad \Delta, B, \Gamma \vdash t : A}{\Delta, \Gamma \vdash t[\Uparrow^i (u/)] : A}$$

We would like to type a term of the form $(\lambda t')u'$, that is

$$\dfrac{\dfrac{B, \Delta, \Gamma \vdash t' : A}{\Delta, \Gamma \vdash \lambda t' : B \to A} \quad \Delta, \Gamma \vdash u' : B}{\Delta, \Gamma \vdash (\lambda t')u' : A}$$

The problem is to build a term t' from t which would be typeable in the environment B, Δ, Γ instead of Δ, B, Γ and a term u' from u which would be typeable in the environment Δ, Γ instead of Γ. This is exactly the work of the functions $I_i(\ )$ and $K_i(\ )$ respectively. Look now at the typing derivation of $t[\Uparrow^i (\uparrow)]$, where $\Delta = D_i, ..., D_1$ $(i = |\Delta|)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{B,\Gamma \vdash \uparrow \rhd \Gamma}{D_1, B, \Gamma \vdash \Uparrow (\uparrow) \rhd D_1, \Gamma}}{\vdots}}{D_{i-1}, ..., D_1, B, \Gamma \vdash \Uparrow^{i-1} (\uparrow) \rhd D_{i-1}, ..., D_1, \Gamma}}{D_i, D_{i-1}, ..., D_1, B, \Gamma \vdash \Uparrow^{i} (\uparrow) \rhd D_i, D_{i-1}, ..., D_1, \Gamma} \quad \Delta, \Gamma \vdash t : A}{\Delta, B, \Gamma \vdash t[\Uparrow^i (\uparrow)] : A}$$

The problem here is to build a term t' from t which would be typeable in the environment Δ, B, Γ instead of Δ, Γ. This is done by the function $J_i(\ )$. We can state the property that should verify those functions.

Property 4.2 For any term t we have (with $i = |\Delta|$) :

- $\Gamma \vdash t : A \quad \Rightarrow \quad \Delta, \Gamma \vdash K_i(t) : A$
- $\Delta, B, \Gamma \vdash t : A \quad \Rightarrow \quad B, \Delta, \Gamma \vdash I_i(t) : A$
- $\Delta, \Gamma \vdash t : A \quad \Rightarrow \quad \Delta, B, \Gamma \vdash J_i(t) : A$

We can then check that the term obtained by the function *Ateb* is typeable.

Lemma 4.3

$$\Gamma \vdash t : A \ \Rightarrow \ \Gamma \vdash Ateb(t) : A$$

Proof: By induction on the typing derivation of t.

- $t = n$ and

$$\dfrac{}{\Gamma, A, \Delta \vdash n : A}$$

We then have $Ateb(t) = n$ and the same typing derivation.

- $t = (u\ v)$ and

$$\dfrac{\Gamma \vdash u : B \rightarrow A \quad \Gamma \vdash v : B}{\Gamma \vdash (u\ v) : A}$$

By induction hypothesis, we have $\Gamma \vdash Ateb(u) : B \rightarrow A$ and $\Gamma \vdash Ateb(v) : B$. We can type $Ateb(t) = Ateb(u)\ Ateb(v)$ as follows

$$\dfrac{\Gamma \vdash Ateb(u) : B \rightarrow A \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash (Ateb(u)\ Ateb(v)) : A}$$

- $t = \lambda u$ and

$$\frac{B, \Gamma \vdash u : A}{\Gamma \vdash \lambda u : B \to A}$$

By induction hypothesis, we have $\Gamma, x : B \vdash Ateb(u) : A$. We can type $Ateb(t) = \lambda Ateb(u)$ as follows

$$\frac{B, \Gamma \vdash Ateb(u) : A}{\Gamma \vdash \lambda Ateb(u) : A}$$

- Cases for $t = u[\Uparrow^i (v/)]$ and $t = u[\Uparrow^i (\uparrow)]$ are treated as discussed above, using property 4.2.

■

Of course, for any t, $Ateb(t)$ does not contain any substitutions.

4.2.1 Functions definitions

The function $J_i(t)$ performs the re-indexing of t as if a substitution $[\Uparrow^i (\uparrow)]$ has been propagated. Since it is applied to terms obtained by the *Ateb* function, only terms without substitutions are concerned.

Here follows its definition:

$$\begin{array}{lcll} J_i(n) & = & n+1 & \text{if } n > i \\ J_i(n) & = & n & \text{if } n \leq i \\ J_i(t\ u) & = & J_i(t)\ J_i(u) & \\ J_i(\lambda t) & = & \lambda J_{i+1}(t) & \end{array}$$

The function $K_i(t)$ performs the re-indexing of t as if i substitutions $[\uparrow]$ have been propagated. We can define it from with the help of the function $J_i(t)$:

$$K_i(t) = \underbrace{J_0(J_0(...J_0(t)))}_{i}$$

When this function is applied to a variable, we obtain $K_i(n) = n + i$.

The function $I_i(t)$ prepares a term t to be applied to a substitution that has lost its $\Uparrow$. It deals also with substitution-free terms.

Here follows its definition:

$$\begin{array}{lcll} I_i(n) & = & n & \text{si } n > i+1 \\ I_i(n) & = & 1 & \text{si } n = i+1 \\ I_i(n) & = & n+1 & \text{si } n \leq i \\ I_i(t\ u) & = & I_i(t)\ I_i(u) & \\ I_i(\lambda t) & = & \lambda I_{i+1}(t) & \end{array}$$

Indices are transformed as follows: since we have deleted $i \Uparrow$, the index $i+1$ must become 1. To reflect this change, every index j smaller than $i+1$ must become $j+1$. The others are let unchanged.

Here follows several useful properties.

Property 4.4 For all t, u, i, j, We have

$$K_i(t) = K_j(u) \qquad \Rightarrow \qquad K_{i+1}(t) = K_{j+1}(u)$$

Proof: Indeed,

$$K_i(t) = K_j(u) \qquad \Rightarrow \qquad J_0(K_i(t)) = J_0(K_j(u))$$

■

Property 4.5 For all n and i, we have

$$J_{i+1}(n) \qquad = \qquad K_1(J_i(n-1))$$

Proof: We calculate the values accordingly to n and i.

- if $n > i+1$ then $J_{i+1}(n) = n$, $J_i(n-1) = n-1$ and $K_1(n-1) = n$.
- if $n \leq i+1$ then $J_{i+1}(n) = n+1$, $J_i(n-1) = n$ and $K_1(n) = n+1$.

■

Property 4.6 For all $n > 1$ and i, we have

$$I_{i+1}(n) \qquad = \qquad J_1(I_i(n-1))$$

Proof: We calculate the values accordingly to n and i.

- if $n > i+2$ then $I_{i+1}(n) = n$, $I_i(n-1) = n-1$ and $J_1(n-1) = n$.
- if $n = i+2$ then $I_{i+1}(n) = 1$, $I_i(n-1) = 1$ and $J_1(1) = 1$.
- if $n < i+2$ then $I_{i+1}(n) = n+1$, $I_i(n-1) = n$ and $J_1(n) = n+1$.

■

Example 4.7 We apply those function to our example, and we obtain

$$J_2(((\lambda t)u)\ ((\lambda I_3(v))K_3(w))) \quad = \quad ((\lambda J_3(t))J_2(u))\ ((\lambda J_3(I_3(v)))J_2(K_3(w)))$$

We can now prove Property 4.2.

Proof:

- $\Delta, \Gamma \vdash t : A \quad \Rightarrow \quad \Delta, B, \Gamma \vdash J_i(t) : A$. By induction on t.
  - $t = n$ with $n \leq i$: $J_i(t) = n$. We have

    $$\frac{}{\Delta_1, A, \Delta_2, \Gamma \vdash n : A}$$

    with $n = |\Delta_1| + 1$. We conclude with the following typing derivation

    $$\frac{}{\Delta_1, A, \Delta_2, B, \Gamma \vdash n : A}$$
  - $t = n$ with $n > i$: $J_i(t) = n + 1$. We have

    $$\frac{}{\Delta, \Gamma_1, A, \Gamma_2 \vdash n : A}$$

    With $n = |\Delta| + |\Gamma_1| + 1$. We the get $n + 1 = |\Delta| + |\Gamma_1| + 1 + 1$ and

    $$\frac{}{\Delta, B, \Gamma_1, A, \Gamma_2 \vdash n : A}$$
  - $t = (u\ v)$: $J_i(t) = (J_i(u)\ J_i(v))$ and we conclude by applying twice the induction hypothesis.
  - $t = \lambda u$ (with $A = C \to D$) : $J_i(t) = \lambda J_{i+1}(u)$. We have

    $$\frac{C, \Delta, \Gamma \vdash u : D}{\Delta, \Gamma \vdash \lambda u : C \to D}$$

    By induction hypothesis, we have $C, \Delta, B, \Gamma \vdash J_{i+1}(u) : D$ and we can build the following typing derivation

    $$\frac{C, \Delta, B, \Gamma \vdash J_{i+1}(u) : D}{\Delta, B, \Gamma \vdash \lambda J_{i+1}(u) : C \to D}$$
- $\Gamma \vdash t : A \quad \Rightarrow \quad \Delta, \Gamma \vdash K_i(t) : A$. By induction hypothesis on t.
  - $t = n$: $K_i(n) = \underbrace{J_0(J_0(...J_0(n)))}_{i} = n + i$. We have

    $$\frac{}{\Gamma_1, A, \Gamma_2 \vdash n : A}$$

    with $n = |\Gamma_1| + 1$. Since $i = |\Delta|$, we get $n + i = |\Gamma_1| + |\Delta| + 1$ and

    $$\frac{}{\Delta, \Gamma_1, A, \Gamma_2 \vdash n + i : A}$$

  - $t = (u\ v)$: $K_i(t) = (K_i(u)\ K_i(v))$ and we conclude by applying twice the induction hypothesis.
  - $t = \lambda u$ (with $A = C \to D$): $K_i(t) = \underbrace{J_0(J_0(...J_0}_{i}(\lambda u))) = \lambda \underbrace{J_1(J_1(...J_1}_{i}(u)))$.

    We get

$$\frac{C, \Gamma \vdash u : D}{\Gamma \vdash \lambda u : C \to D}$$

    By the item above, we have

$$\begin{array}{c} C, \Gamma \vdash u : D \\ \Downarrow \\ C, E_1, \Gamma \vdash J_1(u) : D \\ \Downarrow \\ C, E_2, E_1, \Gamma \vdash J_1(J_1(u)) : D \\ \Downarrow \\ \vdots \\ \Downarrow \\ C, E_i, ..., E_1, \Gamma \vdash \underbrace{J_1(J_1(...J_1}_{i}(u))) \end{array}$$

    with $\Delta = E_i, ..., E_1$. We can then build the following typing derivation

$$\frac{C, \Delta, \Gamma \vdash \underbrace{J_1(J_1(...J_1}_{i}(u))) : D}{\Delta, \Gamma \vdash K_i(\lambda u) : C \to D}$$

- $\Delta, B, \Gamma \vdash t : A \quad \Rightarrow \quad B, \Delta, \Gamma \vdash I_i(t) : A$. By induction on t.
  - $t = n$ with $n > i + 1$: $I_i(t) = n$. We have

$$\frac{}{\Delta, B, \Gamma_1, A, \Gamma_2 \vdash n : A}$$

    with $n = |\Delta| + 1 + |\Gamma_1| + 1$. We conclude with the following typing derivation

$$\frac{}{B, \Delta, \Gamma_1, A, \Gamma_2 \vdash n : A}$$

  - $t = n$ with $n = i + 1$: $I_i(t) = 1$. We have

$$\frac{}{\Delta, B, \Gamma \vdash n : B}$$

with $n = i + 1 = |\Delta| + 1$. We conclude with the following typing derivation

$$\frac{}{B, \Delta, \Gamma \vdash 1 : A}$$

- $t = n$ with $n \leq i$: $I_i(t) = n + 1$. We have

$$\frac{}{\Delta_1, A, \Delta_2, B, \Gamma \vdash n : A}$$

  with $n = |\Delta_1| + 1$. We then get $n + 1 = |\Delta_1| + 1 + 1$ and

$$\frac{}{B, \Delta_1, A, \Delta_2, \Gamma \vdash n : A}$$

- $t = (u\ v)$: $I_i(t) = (I_i(u)\ I_i(v))$ and we conclude by applying twice the induction hypothesis.
- $t = \lambda u$ (with $A = C \to D$) : $I_i(t) = \lambda I_{i+1}(u)$. We have

$$\frac{C, \Delta, B, \Gamma \vdash u : D}{\Delta, B, \Gamma \vdash \lambda u : C \to D}$$

  By induction hypothesis, we have $C, \Delta, \Gamma \vdash I_{i+1}(u) : D$ and we can build the following typing derivation

$$\frac{C, \Delta, \Gamma \vdash I_{i+1}(u) : D}{\Delta, \Gamma \vdash \lambda I_{i+1}(u) : C \to D}$$

■

4.2.2 Definition of the relation $\lessdot$

The function *Ateb* erases the substitutions $[\Uparrow^i (\uparrow)]$ and we will not be able to recover them by reducing the term obtained, as is shown for the following example.

Example 4.8 We continue with our example, we get

$$\begin{array}{c} ((\lambda J_3(t))J_2(u))\ ((\lambda J_3(I_3(v)))J_2(K_3(w))) \\ \to_B \to_B \\ J_3(t)[J_2(u)/]\ J_3(I_3(v))[J_2(K_3(w))/] \end{array}$$

We must use the proof by simulation. To perform this simulation, we need a new function $\bar{t}$ which performs the re-indexing of the erased substitutions.

$$\begin{array}{lcl}
\overline{n} & = & n \\
\overline{t\ u} & = & \overline{t}\ \overline{u} \\
\overline{\lambda t} & = & \lambda \overline{t} \\
\overline{t[u/]} & = & \overline{t}[\overline{u}/] \\
\overline{t[\Uparrow^i (u/)]} & = & I_i(\overline{t})[K_i(\overline{u})/] \\
\overline{t[\Uparrow^i (\uparrow)]} & = & J_i(\overline{t})
\end{array}$$

This function will deal with terms that might contain substitutions. We need then to extend their definition. By the way, since the function $\overline{t}$ removes from t the $\Uparrow$ and $\uparrow$, we will restrain our-self to the simple substitution case:

$$\begin{array}{lcl}
J_i(t[u/]) & = & J_{i+1}(t)[J_i(u)/] \\
I_i(t[u/]) & = & I_{i+1}(t)[I_i(u)/]
\end{array}$$

The function $\bar{\cdot}$ commute with the other function, as stated in the following lemmas.

Lemma 4.9 for all i and t (without $\Uparrow$ and $\uparrow$) we have

$$\overline{J_i(t)} \quad = \quad J_i(\overline{t})$$

Proof: By induction on t.

- If $t = n$, then $J_i(t) = n'$, $\overline{n'} = n'$ on one side, and $\overline{n} = n$ on the other side.
- In all the other cases, we conclude by induction hypothesis.

■

Lemma 4.10 For all i and t (without $\Uparrow$ and $\uparrow$) we have

$$\overline{I_i(t)} \quad = \quad I_i(\overline{t})$$

Proof: By induction on t.

- If $t = n$, then $I_i(t) = n'$, $\overline{n'} = n'$ on one side, and $\overline{n} = n$ on the other side.
- In all the other cases, we conclude by induction hypothesis.

■

Lemma 4.11 For all i and t (without $\Uparrow$ and $\uparrow$) we have

$$\overline{K_i(t)} \quad = \quad K_i(\overline{t})$$

Proof: This is a direct consequence of Lemma 4.9. ■

We can check that this function is correct w.r.t. our example.

Example 4.12 Here is the final term obtain for our example:

$$\begin{array}{c}\overline{J_3(t)[J_2(u)/]\ J_3(I_3(v))[J_2(K_3(w))/]} \\ = \\ J_3(\overline{t})[J_2(\overline{u})/]\ J_3(I_3(\overline{v}))[J_2(K_3(\overline{w}))/]\end{array}$$

Here is the original term:

$$\begin{array}{c}\overline{(t[u/]\ v[\Uparrow(\Uparrow(\Uparrow(w/)))])[\Uparrow(\Uparrow(\uparrow))]} \\ = \\ J_2(\overline{t}[\overline{u}/]\ I_3(\overline{v})[K_3(\overline{w})/]) \\ = \\ J_3(\overline{t})[J_2(\overline{u})/]\ J_3(I_3(\overline{v}))[J_2(K_3(\overline{w}))/]\end{array}$$

We also need an order relation on the skeleton of terms. We want that $t \preccurlyeq t'$ if and only if t contains $[\uparrow]$ and $\Uparrow$ only where t' contains them also. We formalize this definition as follows:

$$\begin{array}{ccc}\text{for all } n \text{ and } m & & n \preccurlyeq m \\ t \preccurlyeq t' \text{ and } u \preccurlyeq u' & \Rightarrow & (t\ u) \preccurlyeq (t'\ u') \\ t \preccurlyeq t' & \Rightarrow & \lambda t \preccurlyeq \lambda t' \\ t \preccurlyeq t' & \Rightarrow & t \preccurlyeq t'[\uparrow] \\ t \preccurlyeq t' \text{ and } s \preccurlyeq s' & \Rightarrow & t[s] \preccurlyeq t'[s']\end{array}$$

$$\begin{array}{ccc} & & \uparrow \preccurlyeq \uparrow \\ t \preccurlyeq t' & \Rightarrow & t/ \preccurlyeq t'/ \\ s \preccurlyeq s' & \Rightarrow & \Uparrow(s) \preccurlyeq \Uparrow(s') \\ s \preccurlyeq s' & \Rightarrow & s \preccurlyeq \Uparrow(s')\end{array}$$

Example 4.13 We have $t[\Uparrow(t'/)] \preccurlyeq t[\uparrow][\Uparrow(\Uparrow(\Uparrow(t'/)))]$.

From this relation and the function $\overline{t}$, we can build a relation to perform our simulation. We note this relation $\lessdot$ and we define it as follows:

$$t \lessdot t' \iff \overline{t} = \overline{t'} \text{ and } t \preccurlyeq t'$$

Remark that we always have $t \lessdot t$. We can now initialize our simulation.

Lemma 4.14 (Initialization) For all t, there exists u such that $Ateb(t) \to_B^* u$ and $u \lessdot t$.

Proof: By induction on t.

- If $t = n$, then $Ateb(t) = n$ and it is enough to take $u = n$.

- If $t = t_1\ t_2$, then $Ateb(t) = Ateb(t_1)\ Ateb(t_2)$. By induction hypothesis, there exists u_1 and u_2 such that $Ateb(t_1) \to_B^* u_1$ and $Ateb(t_2) \to_B^* u_2$ with $u_1 \lessdot t_1$ and $u_2 \lessdot t_2$. We take $u = u_1\ u_2$.

- If $t = \lambda t'$, then we proceed as above using the induction hypothesis for t'.

- If $t = t'[\Uparrow^i (\uparrow)]$, then $Ateb(t) = J_i(Ateb(t'))$. By induction hypothesis, there exists u' such that $Ateb(t') \to_B^* u'$ and $u' \lessdot t'$. We take $u = J_i(u')$ and we check that $u \lessdot t$, that is $\overline{u} = \overline{t}$ and $u \preccurlyeq t$. This last condition is trivial since $u' \preccurlyeq t'$. We calculate $\overline{u} = \overline{J_i(u')}$, which is equal to $J_i(\overline{u'})$ by Lemma 4.9. $\overline{t} = \overline{t'[\Uparrow^i (\uparrow)]} = J_i(\overline{t'})$, and we conclude since $\overline{u'} = \overline{t'}$.

- If $t = t_1[\Uparrow^i (t_2/)]$, then $Ateb(t) = (\lambda I_i(Ateb(t_1)))K_i(Ateb(t_2))$. By induction hypothesis, there exists u_1 and u_2 such that $Ateb(t_1) \to_B^* u_1$ and $Ateb(t_2) \to_B^* u_2$ with $u_1 \lessdot t_1$ and $u_2 \lessdot t_2$. We take $u' = (\lambda I_i(u_1))K_i(u_2)$ for which it is clear that $Ateb(t) \to_B^* u'$. We have $u' \to_B I_i(u_1)[K_i(u_2)/]$, we take this last term as u and we check that $u \lessdot t$, that is $\overline{u} = \overline{t}$ and $u \preccurlyeq t$. This last condition is trivial since $u_1 \preccurlyeq t_1$ and $u_2 \preccurlyeq t_2$. We calculate $\overline{u} = \overline{I_i(u_1)[K_i(u_2)/]}$, which is equal to $I_i(\overline{u_1})[K_i(\overline{u_2})/]$ by Lemmas 4.10 and 4.11. $\overline{t} = \overline{t_1[\Uparrow^i (t_2/)]} = I_i(\overline{t_1})[K_i(\overline{t_2})/]$, and we conclude since $\overline{u_1} = \overline{t_1}$ and $\overline{u_2} = \overline{t_2}$.

■

4.2.3 Simulation Lemmas

We need several lemmas in order to prove the simulation of reductions of $\lambda\upsilon$. We separate the reduction rules in two subset: we call R_1 the set containing the rule B alone and R_2 the set containing all the other rules. Of course, R_2 is strongly normalizing (see [3]). We want to establish the following diagrams:

$$\begin{tikzcd} t \arrow[r, "B"'] & t' \\ \rotatebox{90}{$\lessdot$} & \rotatebox{90}{$\lessdot$} \\ u \arrow[r, "+", "\lambda\upsilon"'] & u' \end{tikzcd} \qquad \begin{tikzcd} t \arrow[r, "R_2"'] & t' \\ \rotatebox{90}{$\lessdot$} & \rotatebox{90}{$\lessdot$} \\ u \arrow[r, "*", "\lambda\upsilon"'] & u' \end{tikzcd}$$

We look first at the simulation of B, then at that of the other.

Lemma 4.15 For all $t \to_B t'$, for all $u \lessdot t$ there exists u' such that $u \to_B u'$ and $u' \lessdot t'$.

$$\begin{tikzcd} t \arrow[r, "B"'] & t' \\ \rotatebox{90}{$\lessdot$} & \rotatebox{90}{$\lessdot$} \\ u \arrow[r, "B"'] & u' \end{tikzcd}$$

Proof: Let $t = (\lambda v)w$ and $(\lambda v)w \to_B v[w/]$, every terms $u \lessdot t$ are of the form $(\lambda v')w'$ with $v' \lessdot v$ and $w' \lessdot w$, we can the reduce $(\lambda v')w' \to_B v'[w'/]$ and the conclusion follows immediately. ■

Lemma 4.16 For all $t \to_{R_2} t'$, for all $u \lessdot t$ there exists u' such that $u \to^*_{\lambda v} u'$ and $u' \lessdot t'$.

$$\begin{tikzcd}
t \arrow[r, "\to_{R_2}", phantom] \arrow[d, phantom, "\rotatebox{90}{$\lessdot$}"] & t' \arrow[d, phantom, "\rotatebox{90}{$\lessdot$}"] \\
u \arrow[r, "\to^*_{\lambda v}", phantom] & u'
\end{tikzcd}$$

Proof: By case on the rule of R_2.

- *FVar*: $1[v/] \to v$. Every terms $u \lessdot 1[v/]$ are of the form $1[v'/]$ with $v' \lessdot v$ and $1[v'/] \to_{FVar} v'$.

- *RVar*: $n + 1[v/] \to n$. Every terms $u \lessdot n + 1[v/]$ are of the form $n + 1[v'/]$ with $v' \lessdot v$ and $n + 1[v'/] \to_{RVar} n$.

- *App*: $t = (v\ w)[s] \to (v[s])\ (w[s]) = t'$. We proceed by case on the form of$s$.
  - if $s = \Uparrow^i (\uparrow)$ then the terms $u \lessdot (v\ w)[\Uparrow^i (\uparrow)]$ might have two distinct forms:
    * either $u = (v'\ w')[\Uparrow^j (\uparrow)]$ with $v' \lessdot v$, $w' \lessdot w$, $j \leq i$ and $\overline{u} = \overline{t}$, that is:

$$\begin{array}{ccc}
\overline{(v'\ w')[\Uparrow^j (\uparrow)]} & = & \overline{(v\ w)[\Uparrow^i (\uparrow)]} \\
= & & = \\
J_j(\overline{v'}\ \overline{w'}) & = & J_i(\overline{v}\ \overline{w}) \\
= & & = \\
J_j(\overline{v'})\ J_j(\overline{w'}) & = & J_i(\overline{v})\ J_i(\overline{w})
\end{array}$$

      which implies $J_j(\overline{v'}) = J_i(\overline{v})$ and $J_j(\overline{w'}) = J_i(\overline{w})$. In that case, $(v'\ w')[\Uparrow^j (\uparrow\ )\ ] \to_{App} (v'[\Uparrow^j (\uparrow)])\ (w'[\Uparrow^j (\uparrow)])$ and we can easily conclude with $(v'[\Uparrow^j (\uparrow)])\ (w'[\Uparrow^j (\uparrow)]) \lessdot (v[\Uparrow^i (\uparrow\ )])\ (w[\Uparrow^i (\uparrow)])$.
    * either $u = (v'\ w')$ with $v' \lessdot v$, $w' \lessdot w$, and $\overline{u} = \overline{t}$, that is:

$$\begin{array}{ccc}
\overline{v'\ w'} & = & \overline{(v\ w)[\Uparrow^i (\uparrow)]} \\
= & & = \\
\overline{v'}\ \overline{w'} & = & J_i(\overline{v}\ \overline{w}) \\
= & & = \\
\overline{v'}\ \overline{w'} & = & J_i(\overline{v})\ J_i(\overline{w})
\end{array}$$

which implies $\overline{v'} = J_i(\overline{v})$ and $\overline{w'} = J_i(\overline{w})$. In that case, $(v'\ w')$ can't be reduce and we can conclude with $(v'\ w') \lessdot (v[\Uparrow^i (\uparrow)])\ (w[\Uparrow^i (\uparrow)])$.

- if $s = \Uparrow^i (r/)$ then all terms $u \lessdot (v\ w)[\Uparrow^i (r/)]$ are of the form $(v'\ w')[\Uparrow^j (r'/)]$ with $v' \lessdot v$, $w' \lessdot w$, $r' \lessdot r$, $j \leq i$ and $\overline{u} = \overline{t}$, that is:

$$\begin{array}{ccc} \overline{(v'\ w')[\Uparrow^j (r'/)]} & = & \overline{(v\ w)[\Uparrow^i (r/)]} \\ = & & = \\ I_j(\overline{v'}\ \overline{w'})[K_j(\overline{r'})/] & = & I_i(\overline{v}\ \overline{w})[K_i(\overline{r})/] \\ = & & = \\ (I_j(\overline{v'})\ I_j(\overline{w'}))[K_j(\overline{r'})/] & = & (I_i(\overline{v})\ I_i(\overline{w}))[K_i(\overline{r})/] \end{array}$$

which implies $J_j(\overline{v'}) = J_i(\overline{v})$, $J_j(\overline{w'}) = J_i(\overline{w})$ and $K_j(\overline{r'}) = K_i(\overline{r})$. In that case, $(v'\ w')[\Uparrow^j (r'/)] \rightarrow_{App} (v'[\Uparrow^j (r'/)])\ (w'[\Uparrow^j (r'/)])$ and we can easily conclude with $(v'[\Uparrow^j (r'/)])\ (w'[\Uparrow^j (r'/)]) \lessdot (v[\Uparrow^i (r/)])\ (w[\Uparrow^i (r/)])$.

- *Lambda*: $t = (\lambda v)[s] \rightarrow \lambda(v[\Uparrow (s)]) = t'$. We proceed by case on the form of s.
  - if $s = \Uparrow^i (\uparrow)$ then the terms $u \lessdot (\lambda v)[\Uparrow^i (\uparrow)]$ might have two distinct forms:
    - either $u = (\lambda v')[\Uparrow^j (\uparrow)]$ with $v' \lessdot v$, $j \leq i$ and $\overline{u} = \overline{t}$, that is :

$$\begin{array}{ccc} \overline{(\lambda v')[\Uparrow^j (\uparrow)]} & = & \overline{(\lambda v)[\Uparrow^i (\uparrow)]} \\ = & & = \\ J_j(\overline{\lambda v'}) & = & J_i(\overline{\lambda v}) \\ = & & = \\ \lambda J_{j+1}(\overline{v'}) & = & \lambda J_{i+1}(\overline{v}) \end{array}$$

which implies $J_{j+1}(\overline{v'}) = J_{i+1}(\overline{v})$. In that case, $(\lambda v')[\Uparrow^j (\uparrow)] \rightarrow_{Lambda} \lambda(v'[\Uparrow^{j+1} (\uparrow)])$ and we can easily conclude with $\lambda(v'[\Uparrow^{j+1} (\uparrow)]) \lessdot \lambda(v[\Uparrow^{i+1} (\uparrow)])$.
    - either $u = \lambda v'$ with $v' \lessdot v$, and $\overline{u} = \overline{t}$ that is:

$$\begin{array}{ccc} \overline{\lambda v'} & = & \overline{(\lambda v)[\Uparrow^i (\uparrow)]} \\ = & & = \\ \lambda\overline{v'} & = & J_i(\overline{\lambda v}) \\ = & & = \\ \lambda\overline{v'} & = & \lambda J_{i+1}(\overline{v}) \end{array}$$

which implies $\overline{v'} = J_{i+1}(\overline{v})$. In that case, $\lambda v'$ can't be reduced and we can conclude with $\lambda v' \lessdot \lambda(v[\Uparrow^{i+1} (\uparrow)])$.

- if $s = \Uparrow^i (r/)$ then all the terms $u \lessdot (\lambda v)[\Uparrow^i (r/)]$ are of the form $(\lambda v')[\Uparrow^j (r'/)]$ with $v' \lessdot v$, $r \lessdot r'$, $j \leq i$ and $\overline{u} = \overline{t}$, that is:

$$\begin{array}{ccc} \overline{(\lambda v')[\Uparrow^j (r'/)]} & = & \overline{(\lambda v)[\Uparrow^i (r/)]} \\ = & & = \\ I_j(\overline{\lambda v'})[K_j(\overline{r'})/] & = & I_i(\overline{\lambda v})[K_i(\overline{r})/] \\ = & & = \\ \lambda I_{j+1}(\overline{v'})[K_j(\overline{r'})/] & = & \lambda I_{i+1}(\overline{v})[K_i(\overline{r})/] \end{array}$$

  which implies $J_{j+1}(\overline{v'}) = J_{i+1}(\overline{v})$ and $K_j(\overline{r'}) = K_i(\overline{r})$. In that case, $(\lambda v')[\Uparrow^j (r'/)] \rightarrow_{Lambda} \lambda(v'[\Uparrow^{j+1} (r'/)])$ and we can easily conclude with $\lambda(v'[\Uparrow^{j+1} (r'/)]) \lessdot \lambda(v[\Uparrow^{i+1} (r/)])$ due to Property 4.4.

- *VarShift*: $n[\uparrow] \rightarrow n+1$. The only two terms $u \lessdot n[\uparrow]$ are $n[\uparrow]$ and $n+1$, we can then conclude with possibly a reduction step using *VarShift*.

- *FVarLift*: $t = 1[\Uparrow (s)] \rightarrow_{FVarLift} 1 = t'$. We proceed by case on the form of s.

  - if $s = \Uparrow^i (\uparrow)$ then the terms $u \lessdot 1[\Uparrow (\Uparrow^i (\uparrow))]$ might have two distinct forms:
    - either $u = 1[\Uparrow (\Uparrow^j (\uparrow))]$ with $j \leq i$ and $\overline{u} = \overline{t}$. We then have $1[\Uparrow (\Uparrow^j (\uparrow))] \rightarrow_{FVarLift} 1$ and we easily conclude.
    - either $u = 1$ with $\overline{u} = \overline{t}$ and we easily conclude.
  - if $s = \Uparrow^i (r/)$ then all the terms $u \lessdot 1[\Uparrow (\Uparrow^i (r/))]$ are of the form $1[\Uparrow (\Uparrow^j (r'/))]$ with $r' \lessdot r$, $j \leq i$ and $\overline{u} = \overline{t}$. We then have $1[\Uparrow (\Uparrow^j (r'/))] \rightarrow_{FVarLift} 1$ and we can conclude.

- *RVarLift*: $t = n+1[\Uparrow (s)] \rightarrow n[s][\uparrow] = t'$. We proceed by case on the form of s.

  - if $s = \Uparrow^i (\uparrow)$ then the terms $u \lessdot n+1[\Uparrow (\Uparrow^i (\uparrow))]$ might have two distinct forms:
    - either $u = n'[\Uparrow^j (\uparrow)]$ with $j \leq i+1$ and $\overline{u} = \overline{t}$, that is:

$$\begin{array}{ccc} \overline{n'+1[\Uparrow (\Uparrow^j (\uparrow))]} & = & \overline{n+1[\Uparrow (\Uparrow^i (\uparrow))]} \\ = & & = \\ J_{j+1}(n') & = & J_{i+1}(n+1) \end{array}$$

      We deduce from this equality that n' can't be smaller than n and that it must then be grater than 1. In that case, $n'[\Uparrow (\Uparrow^j (\uparrow))] \rightarrow_{RVarLift} n'-1[\Uparrow^j (\uparrow)][\uparrow]$ and we must check that

$$\begin{array}{ccc}
\overline{n[\Uparrow^i(\uparrow)][\uparrow]} & = & \overline{n'-1[\Uparrow^j(\uparrow)][\uparrow]} \\
= & & = \\
K_1(J_i(n)) & = & K_1(J_j(n'-1))
\end{array}$$

By Property 4.5, we have $K_1(J_i(n)) = J_{i+1}(n+1)$ and $K_1(J_j(n'-1)) = J_{j+1}(n')$, and we can conclude with $n'-1[\Uparrow^j(\uparrow)][\uparrow] \lessdot n[\Uparrow^i(\uparrow)][\uparrow]$.

$*$ either $u = n'$, and $\overline{u} = \overline{t}$, that is:

$$\begin{array}{ccc}
\overline{n'} & = & \overline{n+1[\Uparrow(\Uparrow^i(\uparrow))]} \\
= & & = \\
n' & = & J_{i+1}(n+1)
\end{array}$$

In that case, u can't be reduced and we must check that

$$\begin{array}{ccc}
\overline{n[\Uparrow^i(\uparrow)][\uparrow]} & = & \overline{n'} \\
= & & = \\
K_1(J_i(n)) & = & n'
\end{array}$$

We conclude with $n' \lessdot n[\Uparrow^i(\uparrow)][\uparrow]$ due to Property 4.5.

– if $s = \Uparrow^i(r/)$ then all the terms $u \lessdot n+1[\Uparrow(\Uparrow^i(r/))]$ are of the form $n'[\Uparrow^j(r'/)]$ with $r \lessdot r'$, $j \leq i$ and $\overline{u} = \overline{t}$, that is:

$$\begin{array}{ccc}
\overline{n'[\Uparrow^j(r'/)]} & = & \overline{n+1[\Uparrow(\Uparrow^i(r/))]} \\
= & & = \\
I_j(n')[K_j(\overline{r'})/] & = & I_{i+1}(n+1)[K_{i+1}(\overline{r})/]
\end{array}$$

which implies $I_j(n') = I_{i+1}(n+1)$ and $K_j(\overline{r'}) = K_{i+1}(\overline{r})$. There are two distinct cases according to the value of j.

$*$ $j = 0$: then we have $I_0(n') = n' = I_{i+1}(n+1)$, $K_0(\overline{r'}) = \overline{r'} = K_{i+1}(\overline{r})$ and we must check that

$$\begin{array}{ccc}
\overline{n[\Uparrow^i(r/)][\uparrow]} & = & \overline{n'[r'/]} \\
= & & \\
K_1(I_i(n)[K_i(r)/]) & & \\
= & & \\
J_0(I_i(n)[K_i(r)/]) & & = \\
= & & \\
J_1(I_i(n))[J_0(K_i(r))/] & & \\
= & & \\
J_1(I_i(n))[K_{i+1}(r)/] & = & n'[\overline{r'}]
\end{array}$$

We can conclude with $J_1(I_i(n)) = I_{i+1}(n+1)n'$ due to Property 4.6.

* $j > 0$: $n'[\Uparrow^j (r'/)]$ reduces to $n'[\Uparrow^{j-1} (r'/)][\uparrow]$ and we must check that

$$\begin{array}{ccc}
\overline{n[\Uparrow^i (r/)][\uparrow]} & = & \overline{n'[\Uparrow^{j-1} (r'/)][\uparrow]} \\
= & & = \\
K_1(I_i(n)[K_i(r)/]) & & K_1(I_{j-1}(n')[K_{j-1}(r')/]) \\
= & & = \\
J_0(I_i(n)[K_i(r)/]) & & J_0(I_{j-1}(n')[K_{j-1}(r')/]) \\
= & & = \\
J_1(I_i(n))[J_0(K_i(r))/] & & J_1(I_{j-1}(n'))[J_0(K_{j-1}(r'))/] \\
= & & = \\
J_1(I_i(n))[K_{i+1}(r)/] & = & J_1(I_{j-1}(n'))[K_j(r')/]
\end{array}$$

and we directly conclude with the help of Property 4.6.

■

4.2.4 Simulation

The function *Ateb* and the relation $\lessdot$ satisfy the hypothesis of Theorem 2.7. We can then apply it and get the desired conclusion.

Corollary 4.17 Since $\lambda\upsilon$-calculus enjoys PSN [3] and simply-typed λ-calculus enjoys SN [11] (which is easily extended to λ-calculus with De Bruijn indices), we have that simply-typed $\lambda\upsilon$-calculus enjoys SN.

5 λ_{wsn}-calculus

In [8] a named version of λ_{ws} was proposed. In current work, we developed a new version of this calculus : λ_{wsn}. We already have a SN proof for this calculus, almost similar to the original one, and this technique can be applied, using the direct proof. We cannot conclude to SN by this way, since PSN has not yet been shown (see [13]).

5.1 Definition

Terms of λ_{wsn}-calculus are given by the following grammar:

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid t[x, t, \Gamma, \Gamma] \mid \Gamma t$$

where Γ is a set of variable. A version of the reduction rules is presented Fig. 2.

Typing rules are given Fig. 3.

$$\begin{array}{llcll}
(b) & (\Delta(\lambda x.t))(\Gamma u) & \to & t[x,u,\Gamma,\Delta] & \\
 & & & & \\
(a) & (t\ u)[x,v,\Gamma,\Delta] & \to & (t[x,v,\Gamma,\Delta]\ u[x,v,\Gamma,\Delta]) & \\
(e_1) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Delta \cup (\Lambda \setminus \{x\}))t & x \in \Lambda \setminus \Gamma \\
(n_1) & y[x,t,\Gamma,\Delta] & \to & \Delta y & x \neq y \text{ or } y \in \Gamma \\
(n_2) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \setminus \Gamma \text{ and } x \notin \Lambda \\
 & \multicolumn{3}{r}{t[y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})]} & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] & \\
 & & & \quad [y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \setminus \Gamma \text{ and } x \notin \Lambda \\
(c_3) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[y,u,(\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] & x \in \Phi \setminus \Gamma \text{ and } x \in \Lambda \\
 & & & & \\
(f) & (\lambda y.t)[x,u,\Gamma,\Delta] & \sim & \lambda y.t[x,u,\Gamma \cup \{y\},\Delta] & \\
(e_2) & (\Lambda t)[x,u,\Gamma,\Delta] & \sim & (\Gamma \cap \Lambda)t[x,u,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] & x \notin \Lambda \setminus \Gamma \\
(d) & \Gamma \Delta t & \sim & (\Gamma \cup \Delta)t & \\
(\emptyset) & \emptyset t & \sim & t & \\
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim & & x \in \Lambda \setminus \Gamma \text{ and } x \notin \Phi \\
 & \multicolumn{3}{r}{t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi]} &
\end{array}$$

Figure 2: Reduction rules of the λ_{wsn}-calculus

5.2 Strong Normalization proof

We define the *Ateb* function as follows:

$$\begin{array}{lcl}
Ateb(x) & = & x \\
Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\
Ateb(\lambda x.t) & = & \lambda x.Ateb(t) \\
Ateb(\Gamma t) & = & \Gamma Ateb(t) \\
Ateb(t[x,u,\Gamma,\Delta]) & = & (\Delta(\lambda x.Ateb(t)))\ (\Gamma Ateb(u))
\end{array}$$

Remark 5.1 The *Ateb* function sends λ_{wsn}-terms to a λ-calculus with explicit weakening.

As for the $\lambda\mathtt{x}$-calculus, the *Ateb* function performs exactly the reverse reduction of the rule b. It is then obvious that if $t' = Ateb(t)$ then $t' \to_b^* t$ and that $Ateb(t)$ does not contain any substitution. We must check that the term we get is typeable.

Lemma 5.2

$$\Gamma \vdash t : A \ \Rightarrow\ \Gamma \vdash Ateb(t) : A$$

Proof: By induction on the typing derivation of t. The only interesting case is that of substitution. We have $t = u[x,v,\Gamma,\Delta]$ and

$$\frac{}{x : A \vdash x : A}\ Ax \qquad \frac{\Gamma \setminus \Delta \vdash t : A \quad \Delta \subset \Gamma}{\Gamma \vdash \Delta t : A}\ Weak$$

$$\frac{\Gamma \vdash t : B\,A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A}\ App \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : B\,A}\ Lamb$$

$$\frac{\Pi \setminus \Gamma \vdash u : A \quad \Pi \setminus \Delta, x : A \vdash t : B \quad (\Gamma \cup \Delta) \subset \Pi}{\Pi \vdash t[x, u, \Gamma, \Delta] : B}\ Sub$$

Figure 3: Typing rules of the λ_{wsn}-calculus

$$\frac{\Pi \setminus \Gamma \vdash v : B \quad \Pi \setminus \Delta, x : B \vdash u : A}{\Pi \vdash u[x, v, \Gamma, \Delta] : A}$$

By induction hypothesis, we have $\Pi \setminus \Delta, x : B \vdash Ateb(u) : A$ and $\Pi \setminus \Gamma \vdash Ateb(v) : B$. We can type $Ateb(t) = (\Delta(\lambda x.Ateb(u)))\ \Gamma Ateb(v)$ as follows

$$\frac{\dfrac{\dfrac{\Pi \setminus \Delta, x : B \vdash Ateb(u) : A}{\Pi \setminus \Delta \vdash \lambda x.Ateb(u) : B \to A}}{\Pi \vdash \Delta(\lambda x.Ateb(u)) : B \to A} \quad \dfrac{\Pi \setminus \Gamma \vdash Ateb(v) : B}{\Pi \vdash \Gamma Ateb(v) : B}}{\Pi \vdash (\Delta(\lambda x.Ateb(u)))\ \Gamma Ateb(v) : A}$$

■

We can directly apply Theorem 2.3. Nevertheless, we cannot get any conclusion since PSN has not yet been shown for this calculus.

6 λ_{ws}-calculus

We deal here with the calculus with De Bruijn indices, and difficulties will arise due to them. More precisely, we won't be able to deal with the typing environment as we did for the $\lambda\upsilon$-calculus. The presence of explicit weakening forbid us to rearrange the typing environment as far as we would do. Here follows the reduction rules (Fig. 4) and typing rules of the λ_{ws}-calculus (Fig. 5) where $|\Gamma| = i$ and $|\Delta| = j$.

At the time of writing, we don't know if it would be possible to apply our technique to this calculus.

7 $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus [1] is a calculus with De Bruijn indices and multiple substitutions, adding difficulties over those already there for the $\lambda\upsilon$-calculus. Our application here is only an exercise since this calculus does not enjoy PSN.

$$
\begin{array}{crcll}
b_1 & (\lambda t u) & \to & [0/u,0]t & \\
b_2 & (\langle k\rangle \lambda t u) & \to & [0/u,k]t & \\
f & [i/u,j]\lambda t & \to & \lambda[i+1/u,j]t & \\
a & [i/u,j](t\ v) & \to & (([i/u,j]t)\ ([i/u,j]v)) & \\
e_1 & [i/u,j]\langle k\rangle t & \to & \langle j+k-1\rangle t & si\ i<k \\
e_2 & [i/u,j]\langle k\rangle t & \to & \langle k\rangle[i-k/u,j]t & si\ i\geq k \\
n_1 & [i/u,j]k & \to & k & si\ i>k \\
n_2 & [i/u,j]i & \to & \langle i\rangle u & \\
n_3 & [i/u,j]k & \to & j+k-1 & si\ i<k \\
c_1 & [i/u,j][k/v,l]t & \to & [k/[i-k/u,j]v,j+l-1]t & si\ k\leq i<k+l \\
c_2 & [i/u,j][k/v,l]t & \to & [k/[i-k/u,j]v,l][i-l+1/u,j]t & si\ i\geq k+l \\
d & \langle i\rangle\langle j\rangle t & \to & \langle i+j\rangle t &
\end{array}
$$

Figure 4: Reduction rules

$$
\frac{}{\Gamma, A, \Delta \vdash i : A}\ \mathit{Axiom}
$$

$$
\frac{B,\Gamma \vdash t : C}{\Gamma \vdash \lambda t : B\, C}\ \mathit{Lambda}
\qquad
\frac{\Gamma \vdash t : B\, A \quad \Gamma \vdash u : B}{\Gamma \vdash (tu) : A}\ \mathit{App}
$$

$$
\frac{\Delta, \Pi \vdash u : A \quad \Gamma, A, \Pi \vdash t : B}{\Gamma, \Delta, \Pi \vdash [i/u,j]t : B}\ \mathit{Subst}
\qquad
\frac{\Delta \vdash t : B}{\Gamma, \Delta \vdash \langle i\rangle t : B}\ \mathit{Weak}
$$

Figure 5: Typing rules

Nevertheless, it reduces the question of SN to that of PSN, *i.e.* if PSN is shown, here already follows a correct proof of SN.

7.1 Definition

Terms of the $\lambda\sigma$-calculus are given by the following grammar:

$$
\begin{array}{l}
t ::= 1 \mid (t\ t) \mid \lambda t \mid t[s] \\
s ::= id \mid \ \uparrow \ \mid t \cdot s \mid s \circ s
\end{array}
$$

As usual, we will add infinitely many integer constants $2, 3, ..., n$ with the convention: $n = 1\underbrace{[\uparrow]...[\uparrow]}_{n-1}$. As usually, we will consider that any term n does not contain substitutions.

Here follows the reduction rules:

$$\begin{array}{lll}
(\lambda t)u & \rightarrow_{B} & t[u \cdot id] \\
 & & \\
(t\ u)[s] & \rightarrow_{App} & (t[s])\ (u[s]) \\
(\lambda t)[s] & \rightarrow_{Lambda} & \lambda(t[1 \cdot (s \circ \uparrow)]) \\
1[id] & \rightarrow_{VarId} & 1 \\
1[t \cdot s] & \rightarrow_{VarCons} & t \\
t[s][s'] & \rightarrow_{Clos} & t[s \circ s'] \\
 & & \\
id \circ s & \rightarrow_{IdL} & s \\
\uparrow \circ id & \rightarrow_{ShiftId} & \uparrow \\
\uparrow \circ (t \cdot s) & \rightarrow_{ShiftCons} & s \\
(t \cdot s) \circ s' & \rightarrow_{Map} & t[s'] \cdot (s \circ s') \\
(s_1 \circ s_2) \circ s_3 & \rightarrow_{Ass} & s_1 \circ (s_2 \circ s_3)
\end{array}$$

Here follows the typing rules:

$$\frac{}{A, \Gamma \vdash 1 : A} \qquad \frac{\Gamma \vdash s \rhd \Gamma' \quad \Gamma' \vdash t : A}{\Gamma \vdash t[s] : A}$$

$$\frac{\Gamma \vdash t : B \rightarrow A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A} \qquad \frac{B, \Gamma \vdash t : A}{\Gamma \vdash \lambda t : B \rightarrow A}$$

$$\frac{}{\Gamma \vdash id \rhd \Gamma} \qquad \frac{}{A, \Gamma \vdash \uparrow \rhd \Gamma}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash s \rhd \Gamma'}{\Gamma \vdash t \cdot s \rhd A, \Gamma'} \qquad \frac{\Gamma \vdash s' \rhd \Gamma'' \quad \Gamma'' \vdash s \rhd \Gamma'}{\Gamma \vdash s \circ s' \rhd \Gamma'}$$

We can give a derived rule for indices $n > 1$, (with $n = |\Gamma| + 1$ and $\Gamma = C_1, ..., C_{n-1}$) :

$$\frac{}{\Gamma, A, \Delta \vdash n : A} \qquad \frac{\Gamma, A, \Delta \vdash \uparrow \rhd C_2, ..., C_{n-1}, A, \Delta \qquad \dfrac{\dfrac{C_{n-1}, A, \Delta \vdash \uparrow \rhd A, \Delta \quad \dfrac{}{A, \Delta \vdash 1 : A}}{\vdots}}{C_2, ..., C_{n-1}, A, \Delta \vdash 1\underbrace{[\uparrow]...[\uparrow]}_{n-2} : A}}{\Gamma, A, \Delta \vdash 1\underbrace{[\uparrow]...[\uparrow]}_{n-1} : A}$$

The substitution back-pushing and some of the functions defined below were strongly inspired by [9].

7.2 Towards strong normalization

We proceed similarly to Section 4.2.

We define the *Ateb* function as follows:

$$\begin{array}{lcl}
Ateb(n) & = & n \\
Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\
Ateb(\lambda t) & = & \lambda Ateb(t) \\
Ateb(t[id]) & = & Ateb(t) \\
Ateb(t[\uparrow]) & = & \mathcal{U}_0^1(Ateb(t)) \\
Ateb(t[s \circ s']) & = & Ateb(t[s][s']) \\
Ateb(t[t' \cdot s]) & = & Ateb((\lambda t)[s])\ Ateb(t')
\end{array}$$

Where $\mathcal{U}_i^j(t)$ is a function that we will define below. The goal of this function is to anticipate the propagation of the substitution $[\uparrow]$ and to perform early re-indexing. To understand its necessity, let us look at the derivation of $t[\uparrow]$.

$$\frac{\overline{B, \Gamma \vdash \uparrow \rhd \Gamma} \quad \Gamma \vdash t : A}{B, \Gamma \vdash t[\uparrow] : A}$$

Example 7.1 For instance, if we suppose that for any tt among t, u, v we have $tt = Ateb(tt)$, then we get

$$Ateb((t[u \cdot id]\ v[1 \cdot 1 \cdot 5 \cdot \uparrow])[\uparrow]) \quad = \quad \mathcal{U}_0^1(((\lambda t)u)\ (((\mathcal{U}_0^1(\lambda\lambda\lambda v)5)1)1))$$

The calculus $\mathcal{U}_0^1(t)$ will then increase by 1 all the free variables of t in order to enable the typing of it in the environment B, Γ. We can therefore state the property that this function must verify.

Property 7.2 For any term t without substitutions we have: $\Gamma \vdash t : A \Rightarrow B, \Gamma \vdash \mathcal{U}_0^1(t) : A$.

It is obvious that for any t, $Ateb(t)$ does not contain any substitutions. We can check that $Ateb(t)$ is typeable.

Lemma 7.3

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash Ateb(t) : A$$

Proof: By induction on t.

- $t = 1$ and

$$\overline{A, \Delta \vdash 1 : A}$$

We then have $Ateb(t) = 1$ and the same typing derivation.

- $t = (u\ v)$ and

$$\frac{\Gamma \vdash u : B \to A \quad \Gamma \vdash v : B}{\Gamma \vdash (u\ v) : A}$$

  By induction hypothesis, we have $\Gamma \vdash \mathit{Ateb}(u) : B \to A$ and $\Gamma \vdash \mathit{Ateb}(v) : B$. We can type $\mathit{Ateb}(t) = \mathit{Ateb}(u)\ \mathit{Ateb}(v)$ as follows

$$\frac{\Gamma \vdash \mathit{Ateb}(u) : B \to A \quad \Gamma \vdash \mathit{Ateb}(v) : B}{\Gamma \vdash (\mathit{Ateb}(u)\ \mathit{Ateb}(v)) : A}$$

- $t = \lambda u$ and

$$\frac{B, \Gamma \vdash u : A}{\Gamma \vdash \lambda u : B \to A}$$

  By induction hypothesis, we have $B, \Gamma \vdash \mathit{Ateb}(u) : A$. We can type $\mathit{Ateb}(t) = \lambda \mathit{Ateb}(u)$ as follows

$$\frac{B, \Gamma \vdash \mathit{Ateb}(u) : A}{\Gamma \vdash \lambda \mathit{Ateb}(u) : B \to A}$$

- $t = u[\mathit{id}]$ and

$$\frac{\Gamma \vdash \mathit{id} \rhd \Gamma \quad \Gamma \vdash u : A}{\Gamma \vdash u[\mathit{id}] : A}$$

  We directly conclude by induction hypothesis.

- $t = u[\uparrow]$ and

$$\frac{B, \Gamma \vdash \uparrow \rhd \Gamma \quad \Gamma \vdash u : A}{B, \Gamma \vdash u[\uparrow] : A}$$

  We conclude by induction hypothesis and by Property 7.2.

- $t = u[v \cdot s]$ and

$$\frac{\dfrac{\Gamma \vdash s \rhd \Gamma' \quad \Gamma \vdash v : B}{\Gamma \vdash v \cdot s \rhd B, \Gamma'} \qquad B, \Gamma' \vdash u : A}{\Gamma \vdash u[v \cdot s] : A}$$

  By induction hypothesis, we have $\Gamma \vdash \mathit{Ateb}((\lambda u)[s]) : B \to A$ and $\Gamma \vdash \mathit{Ateb}(v) : B$. We conclude with the following typing derivation

$$\frac{\Gamma \vdash \mathit{Ateb}((\lambda u)[s]) : B \to A \quad \Gamma \vdash \mathit{Ateb}(v) : B}{\Gamma \vdash \mathit{Ateb}((\lambda u)[s])\ \mathit{Ateb}(v) : A}$$

- $t = u[s \circ s']$, and we conclude directly by induction hypothesis.

■

7.2.1 Function definition

The function $\mathcal{U}_i^j(t)$ performs a re-indexing of the term t as if we had propagated a substitution $[\uparrow]$. Since it deals only with terms obtained from the *Ateb* function, we might consider only substitution-free terms. However, we will need later to use it with terms with substitutions, but without $\uparrow$. When it is applied to a substitution it returns a pair composed by an integer and a substitution, else it returns a term.

Here follows its complete definition:

$$\begin{array}{lcll}
\mathcal{U}_i^j(n) & = & n+j & \text{if } n > i \\
\mathcal{U}_i^j(n) & = & n & \text{if } n \leq i \\
\mathcal{U}_i^j(t\ u) & = & \mathcal{U}_i^j(t)\ \mathcal{U}_i^j(u) & \\
\mathcal{U}_i^j(\lambda t) & = & \lambda \mathcal{U}_{i+1}^j(t) & \\
 & & & \\
\mathcal{U}_i^j(t[s]) & = & \text{let } i', s' = \mathcal{U}_i^j(s) & \\
 & & \text{in } \mathcal{U}_{i'}^j(t)[s'] & \\
\mathcal{U}_i^j(id) & = & i, id & \\
\mathcal{U}_i^j(t \cdot s) & = & \text{let } i', s' = \mathcal{U}_i^j(s) & \\
 & & \text{in } i'+1, \mathcal{U}_i^j(t) \cdot s' & \\
\mathcal{U}_i^j(s_1 \circ s_2) & = & \text{let } i_2', s_2' = \mathcal{U}_i^j(s_2) & \\
 & & \text{and } i_1', s_1' = \mathcal{U}_{i_2'}^j(s_1) & \\
 & & \text{in } i_1', s_1' \circ s_2' &
\end{array}$$

The modification of the index i (and the value of the integer part of the pair) reflects the number of $\cdot$ we got through, each of them acting like a λ.

Here follows the proof of Property 7.2.

Proof: We have to proof that, for any t substitution-free, $\Gamma \vdash t : A \Rightarrow B, \Gamma \vdash \mathcal{U}_0^1(t) : A$. Actually we prove a more general result, namely $\Gamma, \Delta \vdash t : A \Rightarrow \Gamma, B, \Delta \vdash \mathcal{U}_i^1(t) : A$ where $i = |\Gamma|$. We proceed by induction on t.

- $t = n$ with $n \leq i$: $\mathcal{U}_i^1(t) = n$. We have

$$\frac{}{\Gamma_1, A, \Gamma_2, \Delta \vdash n : A}$$

with $n = |\Gamma_1| + 1$. We conclude with the following typing derivation

$$\frac{}{\Gamma_1, A, \Gamma_2, B, \Delta \vdash n : A}$$

- $t = n$ with $n > i$: $\mathcal{U}_i^1(t) = n + 1$. On a

$$\frac{}{\Gamma, \Delta_1, A, \Delta_2 \vdash n : A}$$

With $n = |\Gamma| + |\Delta_1| + 1$. We conclude with the following typing derivation

$$\frac{}{\Gamma, B, \Delta_1, A, \Delta_2 \vdash n + 1 : A}$$

- $t = (u\ v)$: $\mathcal{U}_i^1(t) = (\mathcal{U}_i^1(u)\ \mathcal{U}_i^1(v))$. We conclude with twice the induction hypothesis.

- $t = \lambda u$ (with $A = C \to D$) : $\mathcal{U}_i^1(t) = \lambda \mathcal{U}_{i+1}^1(u)$. We have

$$\frac{C, \Gamma, \Delta \vdash u : D}{\Gamma, \Delta \vdash \lambda u : C \to D}$$

By induction hypothesis, we have $C, \Gamma, B, \Delta \vdash \mathcal{U}_{i+1}^1(u) : D$, and we conclude with the following typing derivation

$$\frac{C, \Gamma, B, \Delta \vdash \mathcal{U}_{i+1}^1(u) : D}{\Gamma, B, \Delta \vdash \lambda \mathcal{U}_{i+1}^1(u) : C \to D}$$

■

Here follows a property used below.

Property 7.4 For all t, i, j, l, we have

$$\mathcal{U}_i^j(\mathcal{U}_i^l(t)) \quad = \quad \mathcal{U}_i^{j+l}(t)$$

Proof: By easy induction on t. ■

Example 7.5 We can apply this function to our example, giving

$$\mathcal{U}_0^1(((\lambda t)u)\ (\lambda((\lambda(\mathcal{U}_0^1(\lambda v)w))1))1) \quad = ((\lambda \mathcal{U}_1^1(t))\mathcal{U}_0^1(u))\ ((((\lambda\lambda\lambda \mathcal{U}_3^2(v))6)2)2)$$

7.2.2 Definition of the relation $\lessdot$

The function *Ateb* applied to a term t returns a new term t' that usually cannot be reduce to t. Indeed, the $\uparrow$ disappears and the information they carried is already propagated in t'. The reducts of t' won't have those terms as it is shown in the following example.

Example 7.6 With our last example, we have

$$\begin{array}{c}(\underline{(\lambda\mathcal{U}_1^1(t))\mathcal{U}_0^1(u)})\ ((((\lambda\lambda\lambda\mathcal{U}_3^2(v))6)2)2)\\ \to^*\\ \mathcal{U}_1^1(t)[\mathcal{U}_0^1(u)\cdot id]\ \mathcal{U}_3^2(v)[2\cdot 2\cdot 6\cdot id]\end{array}$$

Remark that the re-indexing of the $\uparrow$ in the original term has correctly been propagated, the substitution $[1\cdot 1\cdot 5\cdot \uparrow]$ has become $[2\cdot 2\cdot 6\cdot id]$.

We now have to simulate the reduction of the initial term by that of the obtained term. We start naively with the following definition which will appear to be inadequate. We will then present an adequate solution.

To perform the simulation, we define a new function $\bar{t}$ that flattens all the re-indexing required in a term t and deletes the lonely substitutions $[id]$.

$$\begin{array}{lcl}
\overline{n} & = & n\\
\overline{t\ u} & = & \bar{t}\ \overline{u}\\
\overline{\lambda t} & = & \lambda\bar{t}\\
\overline{t[s]} & = & \text{let } n, s' = \overline{s} \text{ in}\\
 & & \mathcal{U}_0^n(\bar{t})[s'] \text{ if } s' \neq \emptyset\\
 & & \mathcal{U}_0^n(\bar{t}) \text{ else}\\
\overline{\uparrow} & = & 1, \emptyset\\
\overline{id} & = & 0, \emptyset\\
\overline{t\cdot s} & = & \text{let } n, s' = \overline{s} \text{ in}\\
 & & n, \bar{t}[s'] \text{ if } s' \neq \emptyset\\
 & & n, \bar{t}\cdot id \text{ else}\\
\overline{s_1 \circ s_2} & = & \text{let } n_1, s_1' = \overline{s_1}\\
 & & \text{and } n_2, s_2' = \overline{s_2} \text{ in}\\
 & & n_1+n_2, \emptyset \text{ if } s_1' = s_2' = \emptyset\\
 & & n_1+n_2, \mathcal{U}_0^{n_2}(s_1') \text{ if } s_2' = \emptyset\\
 & & n_1+n_2, s_2' \text{ if } s_1' = \emptyset\\
 & & n_1+n_2, \mathcal{U}_0^{n_2}(s_1')\circ s_2' \text{ else}
\end{array}$$

The function $\bar{\cdot}$ commutes with $\mathcal{U}_i^j(t)$, as stated in the following lemma.

Lemma 7.7 For all i, j and t (without $\uparrow$) we have

$$\overline{\mathcal{U}_i^j(t)} \quad = \quad \mathcal{U}_i^j(\bar{t})$$

Proof: By induction on t.

- If $t = n$, then $\mathcal{U}_i^j(t) = n'$, $\overline{n'} = n'$ and $\overline{n} = n$.
- All the remaining cases are easily proved by induction hypothesis.

■

Example 7.8 Look at the final example term:

$$\begin{array}{c}\overline{\mathcal{U}_1^1(t)[\mathcal{U}_0^1(u)\cdot id]\ \mathcal{U}_3^2(v)[2\cdot 2\cdot 6\cdot id]}\\ =\\ \mathcal{U}_1^1(\overline{t})[\mathcal{U}_0^1(\overline{u})\cdot id]\ \mathcal{U}_3^2(\overline{v})[2\cdot 2\cdot 6\cdot id]\end{array}$$

And the original term:

$$\begin{array}{c}\overline{(t[u\cdot id]\ v[1\cdot 1\cdot 5\cdot\uparrow])[\uparrow]}\\ =\\ \mathcal{U}_0^1(\mathcal{U}_0^0(\overline{t})[\overline{u}\cdot id]\ \mathcal{U}_0^1(\overline{v})[1\cdot 1\cdot 5\cdot id])\\ =\\ \mathcal{U}_1^1(\overline{t})[\mathcal{U}_0^1(\overline{u})\cdot id]\ \mathcal{U}_3^2(\overline{v})[2\cdot 2\cdot 6\cdot id]\end{array}$$

We need an order relation on the skeleton of terms. We want that $t \preccurlyeq t'$ if and only if t' does contain $\uparrow$ and $[id]$ only at the same place t does. More formally, it can be defined as follows:

$$\begin{array}{ccc}
\text{pour tout } n \text{ and } m & & n\preccurlyeq m\\
t\preccurlyeq t' \text{ and } u\preccurlyeq u' & \Rightarrow & (t\ u)\preccurlyeq (t'\ u')\\
t\preccurlyeq t' & \Rightarrow & \lambda t\preccurlyeq \lambda t'\\
t\preccurlyeq t' & \Rightarrow & t\preccurlyeq t'[\uparrow]\\
t\preccurlyeq t' & \Rightarrow & t\preccurlyeq t'[id]\\
t\preccurlyeq t' \text{ and } s\preccurlyeq s' & \Rightarrow & t[s]\preccurlyeq t'[s']
\end{array}$$

$$\begin{array}{ccc}
 & & \uparrow\preccurlyeq\uparrow\\
 & & id\preccurlyeq id\\
 & & id\preccurlyeq\uparrow\\
t\preccurlyeq t' \text{ and } s\preccurlyeq s' & \Rightarrow & t\cdot s\preccurlyeq t'\cdot s'\\
s\preccurlyeq s' & \Rightarrow & s\preccurlyeq s'\circ id\\
s\preccurlyeq s' & \Rightarrow & s\preccurlyeq id\circ s'\\
s\preccurlyeq s' & \Rightarrow & s\preccurlyeq s'\circ\uparrow\\
s\preccurlyeq s' & \Rightarrow & s\preccurlyeq\uparrow\circ s'\\
s_1\preccurlyeq s_1' \text{ and } s_2\preccurlyeq s_2' & \Rightarrow & s_1\circ s_2\preccurlyeq s_1'\circ s_2'
\end{array}$$

Example 7.9 We have $t[t'\cdot id]\preccurlyeq t[id][t'\cdot\uparrow]$.

With this relation and the function $\overline{t}$, we can define a relation to perform our simulation. We denote this relation $\lessdot$ and we define it as follows:

$$t\lessdot t' \iff \overline{t}=\overline{t'} \text{ and } t\preccurlyeq t'$$

We remark that we always have $t\lessdot t$.

However, we cannot go further because this relation will not be adequate to perform the simulation. Indeed, a problem arise to simulate the rule *Abs*: $(\lambda t)[s]\to\lambda(t[1\cdot(s\circ\uparrow)]$. If $s=id$ (or $\uparrow$), then a term $u\lessdot(\lambda t)[id]$ can be λt

which don't verify $\lambda t \lessdot \lambda(t[1\cdot(s\circ\uparrow)]$. We would like to extend our relation to this kind of id (and similarly for $\uparrow$). We start over again with a new relation $\lessdot$ that take this into account.

To solve the problem, we choose to identify terms having the same σ-normal form. We call σ the set of rules without B. The σ-normal form of a term is given by the transitive closure of σ. We now that such a normal form exists since σ is strongly normalizing (see [1]). We denote $\sigma(t)$ the σ-normal form of t.

We first define a notion of *redexability* of terms. The idea is to find all the "bad" terms, that is those that can give rise to B-redices.

Definition 7.10 We say that a term is potentially redexable (denoted $PR(t)$) if it contains application or λ at some node.

We define then the relation $\preccurlyeq$ that will ensure that if $u \preccurlyeq t$ then u has the same redexability than t.

$$\begin{array}{ccll}
\text{pour tout } n \text{ and } m & & n \preccurlyeq m & \\
t \preccurlyeq t' \text{ and } u \preccurlyeq u' & \Rightarrow & (t\ u) \preccurlyeq (t'\ u') & \\
t \preccurlyeq t' & \Rightarrow & \lambda t \preccurlyeq \lambda t' & \\
t \preccurlyeq t' & \Rightarrow & t \preccurlyeq t'[s] & \text{if } \neg PR(s) \\
t \preccurlyeq t' \text{ and } s \preccurlyeq s' & \Rightarrow & t[s] \preccurlyeq t'[s'] & \\
& & & \\
& & \uparrow \preccurlyeq \uparrow & \\
& & id \preccurlyeq id & \\
& & id \preccurlyeq s & \text{if } \neg PR(s) \\
t \preccurlyeq t' \text{ and } s \preccurlyeq s' & \Rightarrow & t\cdot s \preccurlyeq t'\cdot s' & \\
s \preccurlyeq s' & \Rightarrow & s \preccurlyeq s' \circ s_1 & \text{if } \neg PR(s_1) \\
s \preccurlyeq s' & \Rightarrow & s \preccurlyeq s_1 \circ s' & \text{if } \neg PR(s_1) \\
s_1 \preccurlyeq s_1' \text{ and } s_2 \preccurlyeq s_2' & \Rightarrow & s_1 \circ s_2 \preccurlyeq s_1' \circ s_2' &
\end{array}$$

Example 7.11 We have $t[t'\cdot id] \preccurlyeq t[1\cdot id][t'\cdot\uparrow]$.

We define the relation $\lessdot$ as follows.

Definition 7.12 For all t and u, $u \lessdot t \iff u \preccurlyeq t$ and $\sigma(t) = \sigma(u)$.

Remark that we always have $t \lessdot t$.

Here follows several lemmas that will be used to prove the initialization lemma. The first one says that the σ-normal form does not change when one deletes a substitution $[id]$.

Lemma 7.13 For all t, we have $\sigma(t) = \sigma(t[id])$.

Proof: See [1]. ■

Lemma 7.14 For all t, we have $\sigma(\mathcal{U}_0^1(t)) = \mathcal{U}_0^1(\sigma(t))$.

Proof: Since the application of $\mathcal{U}_0^1()$ changes only the values of free variable, its application is orthogonal to the reduction of substitutions that changes only the values of bound variables. ■

The following lemma says that the σ-normal form of a term $t[\uparrow]$ is the same as that of $\mathcal{U}_0^1(t)$.

Lemma 7.15 For all t, we have $\sigma(\mathcal{U}_0^1(t)) = \sigma(t[\uparrow])$.

Proof: We prove a more general result. Let $\Uparrow(s) = 1 \cdot (s \circ \uparrow)$, we prove that for all t and i, we have $\sigma(\mathcal{U}_i^1(t)) = \sigma(t[\Uparrow^i(\uparrow)])$. Since $\sigma(t[\Uparrow^i(\uparrow)]) = \sigma(\sigma(t)[\Uparrow^i(\uparrow)])$, it is enough to prove it for t in σ-normal form. We proceed by induction on it.

- $t = u\ v$: then $\sigma((u\ v)[\Uparrow^i(\uparrow)]) = \sigma((u[\Uparrow^i(\uparrow)])\ \sigma(v[\Uparrow^i(\uparrow)])$, and we conclude by induction hypothesis.
- $t = \lambda u$: then $\sigma((\lambda u)[\Uparrow^i(\uparrow)]) = \lambda(\sigma(u[\Uparrow^{i+1}(\uparrow)]))$ and $\sigma(\mathcal{U}_i^1(\lambda u)) = \lambda(\sigma(\mathcal{U}_{i+1}^1(u)))$. We conclude by induction hypothesis.
- $t = 1$: there are two cases,
  - either $i = 0$, then $\sigma(\mathcal{U}_0^1(1)) = \sigma(2)$ and $\sigma(1[\uparrow]) = \sigma(2)$.
  - or $i > 0$, then $\sigma(\mathcal{U}_i^1(1)) = \sigma(1) = 1$ and $\sigma(1[\Uparrow^i(\uparrow)]) = \sigma(1[1 \cdot (\Uparrow^{i-1}(\uparrow) \circ \uparrow)]) =_{VarCons} \sigma(1) = 1$.
- $t = n > 1$: there are two cases,
  - either $i < n$, then $\sigma(\mathcal{U}_i^1(n)) = \sigma(n+1) = \sigma(1\underbrace{[\uparrow]...[\uparrow]}_{n}) =_{Clos}$ $1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n}]$ and

    $$\sigma(n[\Uparrow^i(\uparrow)]) = \sigma(1\underbrace{[\uparrow]...[\uparrow]}_{n-1}[\Uparrow^i(\uparrow)]) =_{Clos} \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1} \circ \Uparrow^i(\uparrow)])$$

    We prove that this last term is equal to $1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n}]$ by induction on i:
    * $i = 0$: $\sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1} \circ \uparrow]) = 1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n}]$
    * $i > 0$:

    $$\sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1} \circ \Uparrow^i(\uparrow)]) = \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1} \circ (1 \cdot (\Uparrow^{i-1}(\uparrow) \circ \uparrow))]) = \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow) \circ \uparrow])$$

By rule *Clos*, we have $\sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow) \circ \uparrow]) = \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)][\uparrow]) = \sigma(\sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)])[\uparrow])$. Since $i < n$, then $i-1 < n-1$ and we can apply the induction hypothesis on i, giving us $\sigma(\sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)])[\uparrow]) = \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1}][\uparrow]) =_{Clos} 1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n}]$.

– either $i \geq n$, then $\sigma(\mathcal{U}_i^1(n)) = \sigma(n) = \sigma(1\underbrace{[\uparrow]...[\uparrow]}_{n-1}) =_{Clos} 1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1}]$

and

$$\sigma(n[\Uparrow^i(\uparrow)]) = \sigma(1\underbrace{[\uparrow]...[\uparrow]}_{n-1}[\Uparrow^i(\uparrow)]) =_{Clos} \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1} \circ \Uparrow^i(\uparrow)])$$

We prove that this last term is equal to $1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1}]$ by induction on i:

* $i = 0$: impossible since $i \geq n > 1$.
* $i = 1$: impossible since $i \geq n > 1$.
* $i = 2$: we must have $n = 2 = 1[\uparrow]$, giving us

$$\begin{array}{lll} \sigma(1[\uparrow \circ \Uparrow^2(\uparrow)]) & = & \sigma(1[\uparrow \circ (1 \cdot ((1 \cdot (\uparrow \circ \uparrow)) \circ \uparrow))]) \\ & =_{ShiftCons} & \sigma(1[(1 \cdot (\uparrow \circ \uparrow)) \circ \uparrow]) \\ & =_{Map} & \sigma(1[1[\uparrow] \cdot (\uparrow \circ \uparrow \circ \uparrow)]) \\ & =_{VarCons} & 1[\uparrow] \end{array}$$

* $i > 2$:

$$\sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1} \circ \Uparrow^i(\uparrow)]) = \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1} \circ (1 \cdot (\Uparrow^{i-1}(\uparrow) \circ \uparrow))]) = \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow) \circ \uparrow])$$

By rule *Clos*, we have $\sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow) \circ \uparrow]) = \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)][\uparrow])$. Since $i \geq n$, then $i-1 \geq n-1$ and we can apply the induction hypothesis on i, giving us $\sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)][\uparrow]) = \sigma(1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-2}][\uparrow]) =_{Clos} 1[\underbrace{\uparrow \circ ... \circ \uparrow}_{n-1}]$.

■

We also need a lemma to equalize σ-normal forms.

Lemma 7.16 For all t, u, s and s' in σ-normal form, if $\sigma(t[1 \cdot (s \circ \uparrow)]) = \sigma(t'[1 \cdot (s' \circ \uparrow)])$ then $\sigma(t[u \cdot s]) = \sigma(t'[u \cdot s'])$.

Proof: Easy induction on t. ■

We can now prove our initialization lemma.

Lemma 7.17 (Initialisation) For all t, there exists u such that $Ateb(t) \to_B^* u$ and $u \lessdot t$.

Proof: By induction on the number of reduction steps of $Ateb(t)$ and by case analysis of t.

- If $t = n$, then $Ateb(t) = n$ and we conclude with $u = n$.
- If $t = (t_1\ t_2)$, then $Ateb(t) = (Ateb(t_1)\ Ateb(t_2))$. By induction hypothesis, there exists u_1 and u_2 such that $Ateb(t_1) \to_B^* u_1$ and $u_1 \lessdot t_1$ and $Ateb(t_2) \to_B^* u_2$ and $u_2 \lessdot t_2$. We conclude with $u = (u_1\ u_2)$.
- If $t = \lambda t'$, then $Ateb(t) = \lambda Ateb(t')$. By induction hypothesis, there exists u' such that $Ateb(t') \to_B^* u'$. We conclude with $u = \lambda u'$.
- If $t = t'[id]$, then $Ateb(t) = Ateb(t')$. By induction hypothesis, there exists u' such that $Ateb(t') \to_B^* u'$. We take $u = u'$ and we conclude with the help of Lemma 7.13.
- If $t = t'[\uparrow]$, then $Ateb(t) = \mathcal{U}_0^1(Ateb(t'))$. By induction hypothesis, there exists u' such that $Ateb(t') \to_B^* u'$. We take $u = \mathcal{U}_0^1(u')$ and we conclude with the help of Lemmas 7.14 and 7.15.
- If $t = t'[s \circ s']$, then $Ateb(t) = Ateb(t'[s][s'])$. By hypothesis, there exists u' such $Ateb(t'[s][s']) \to_B^* u'$. Four cases arise with respect to the values of $PR(s)$ and $PR(s')$, in all those cases, we can conclude with $u = u'$.
- If $t = t_1[t_2 \cdot s]$, then $Ateb(t) = Ateb((\lambda t_1)[s])\ Ateb(t_2)$. By induction hypothesis, there exists u_1 and u_2 such that $Ateb((\lambda t_1)[s]) \to_B^* u_1$ and $u_1 \lessdot (\lambda t_1)[s]$ and $Ateb(t_2) \to_B^* u_2$ and $u_2 \lessdot t_2$. There are two cases with respect to the form of u_1.
  - If $u_1 = \lambda v_1$ (and so $\neg PR(s)$), then we take $u = v_1[u_2 \cdot id]$. We must check that $u \lessdot t$ and the difficulty resides in the proof of $\sigma(u) = \sigma(t)$. By hypothesis, we have $\sigma(\lambda v_1) = \sigma((\lambda t_1)[s])$. It is obvious that $\sigma(\lambda v_1) = \sigma((\lambda v_1)[id]) = \lambda(\sigma(\sigma(v_1)[1 \cdot (id \circ \uparrow)]))$. On the other hand, we have $\sigma((\lambda t_1)[s]) = \lambda(\sigma(\sigma(t_1)[1 \cdot (\sigma(s) \circ \uparrow)]))$, and it gives us $\sigma(\sigma(v_1)[1 \cdot (id \circ \uparrow)]) = \sigma(\sigma(t_1)[1 \cdot (\sigma(s) \circ \uparrow$

)]). We have the required conditions to apply Lemma 7.16 with the term $\sigma(u_2)$ that is equal to $\sigma(t_2)$ by hypothesis, and we get $\sigma(\sigma(v_1)[\sigma(u_2)\cdot id]) = \sigma(\sigma(t_1)[\sigma(t_2)\cdot s])$ which concludes this point.

– If $u_1 = (\lambda v_1)[s_1]$, then we take $u = v_1[u_2 \cdot s_1]$ and we conclude similarly to the previous point with the help of Lemma 7.16.

■

7.2.3 Simulation

We can now perform the simulation.

Lemma 7.18 (Simulation) For all t reducing by rule B to t', for all $u \lessdot t$, there exists u' such that u reduces in one step to u' and $u' \lessdot t'$. For all he other rules, for all t reducing to t', for all $u \lessdot t$, there exists u' such that u reduces in zero or some steps to u' and $u' \lessdot t'$.

Proof: For all the rules apart from B (*i.e.* σ), the proof is simple. $u \lessdot t$ gives us $u \preccurlyeq t$ and $\sigma(u) = \sigma(t)$, and, on the other hand, $t \rightarrow_\sigma t'$ implies $\sigma(t) = \sigma(t')$. Two cases are possible with respect to the fact that the redex appears also in u. If not, we take $u' = u$ and we directly conclude. Else, we reduce it with the same rule and we conclude with $\sigma(u') = \sigma(u) = \sigma(t')$.

It's more complicated for the rule B. The hypothesis is the same but we are sure that the redex appears in u, that was the point of defining the relation $\preccurlyeq$ with the help of the predicate PR. We then have $u \rightarrow_B u'$ and we want to prove $u' \lessdot t'$. Even if it is obvious that $u' \preccurlyeq t'$ comes directly from $u \preccurlyeq t$, it is not the case for the equality of the σ-normal forms. We want $\sigma(u') = \sigma(t')$ with the hypothesis $\sigma(u) = \sigma(t)$. We take $t = C[(\lambda v)\ w]$, which gives us $t' = C[v[w \cdot id]]$ and $u = C'[(\lambda v')\ w']$. Two cases are possible:

- the redex $(\lambda v)\ w$ does not appear in $\sigma(t)$. It means that the calculus of $\sigma(t)$ can be split as follows:

$$C[(\lambda v)\ w] \rightarrow_\sigma^* C_1[\uparrow \circ (C_2[(\lambda v)\ w] \cdot s)] \rightarrow_{ShiftCons} C_1[s] \rightarrow_\sigma^* \sigma(t)$$

  Since $\sigma(t) = \sigma(u)$, the same occurs for u. Similarly for the redex, the reduct will be erased from t' and from u' and we get $\sigma(u') = \sigma(t')$.

- the redex $(\lambda v)\ w$ does appear in $\sigma(t)$. We will write, for all t, $\underline{\mathrm{t}}$ for $\sigma(t)$, in order to clarify the presentation of the calculi. We have the following equalities:

$$\begin{array}{rcl} \underline{\mathrm{t}} & = & \sigma(C[(\lambda v)\ w]) \\ & = & C_1[\sigma(((\lambda v)\ w)[\underline{\mathrm{s}}])] \\ & = & C_1[(\lambda \sigma(\underline{\mathrm{v}}[1 \cdot (\underline{\mathrm{s}} \circ \uparrow)]))\ \sigma(\underline{\mathrm{w}}[\underline{\mathrm{s}}])] \end{array}$$

And, similarly, $\underline{\text{u}} = C'_1[(\lambda\sigma(\underline{\text{v'}}[1 \cdot (\underline{\text{s'}}\circ \uparrow)]))\ \sigma(\underline{\text{w'}}[\underline{\text{s'}}])]$. From $\underline{\text{t}} = \underline{\text{u}}$ we deduce $C_1 = C'_1$, $\sigma(\underline{\text{v}}[1 \cdot (\underline{\text{s}}\circ \uparrow)]) = \sigma(\underline{\text{v'}}[1 \cdot (\underline{\text{s'}}\circ \uparrow)])$ and $\sigma(\underline{\text{w}}[\underline{\text{s}}]) = \sigma(\underline{\text{w'}}[\underline{\text{s'}}])$. We now look at $\underline{\text{t'}}$ and $\underline{\text{u'}}$:

$$\begin{array}{rl} \underline{\text{t'}} = & \sigma(C[v[w \cdot id]]) \\ = & C_1[\sigma(\underline{\text{v}}[\underline{\text{w}} \cdot id][\underline{\text{s}}])] \\ =_{Clos} & C_1[\sigma(\underline{\text{v}}[(\underline{\text{w}} \cdot id) \circ \underline{\text{s}}])] \\ =_{Map} & C_1[\sigma(\underline{\text{v}}[\underline{\text{w}}[\underline{\text{s}}] \cdot \underline{\text{s}}])] \\ = & C_1[\sigma(\underline{\text{v}}[\sigma(\underline{\text{w}}[\underline{\text{s}}]) \cdot \underline{\text{s}}])] \end{array}$$

And similarly, $\underline{\text{u'}} = C'_1[\sigma(\underline{\text{v'}}[\sigma(\underline{\text{w'}}[\underline{\text{s'}}]) \cdot \underline{\text{s'}}])]$. From the preceding equalities, we deduce $\underline{\text{u'}} = C'_1[\sigma(\underline{\text{v'}}[\sigma(\underline{\text{w}}[\underline{\text{s}}]) \cdot \underline{\text{s'}}])]$, and we can conclude with the help of Lemma 7.16.

■

Lemma 7.19 For all terms t, if $u \lessdot t$ and $u \in \Lambda^X_{SN}$, then $t \in \Lambda^X_{SN}$.

Proof: By the simulation Lemma 7.18, and since the σ-calculus is terminating [1], if we have an infinite derivation of t, then we can also build one in u, and that gives us a contradiction. ■

Since the $Ateb(t)$ function returns a term t' that reduces to $u \lessdot t$ (by Lemma 7.17), we know this technique can be applied to this calculus.

8 $\lambda_{\sigma n}$-calculus

In this section, we study a version with names of the $\lambda\sigma$-calculus [1]. The same remarks will apply here as regards to the application of this technique.

8.1 Definition

Terms of the $\lambda_{\sigma n}$-calculus are given by the following grammar:

$$\begin{array}{l} t ::= x \mid (t\ t) \mid \lambda x.t \mid t[s] \\ s ::= id \mid (t/x) \cdot s \mid s \circ s \end{array}$$

Here follows the reduction rules:

$$
\begin{array}{llll}
(\lambda x.t)u & \to_B & t[(u/x)\cdot id] & \\
\\
(t\ u)[s] & \to_{App} & (t[s])\ (u[s]) & \\
(\lambda x.t)[s] & \to_{Lambda} & \lambda y.(t[(y/x)\cdot s]) & \text{with } y \text{ fresh} \\
x[id] & \to_{VarId} & x & \\
x[(t/x)\cdot s] & \to_{VarCons1} & t & \\
x[(t/y)\cdot s] & \to_{VarCons2} & x[s] & (x \neq y) \\
t[s][s'] & \to_{Clos} & t[s\circ s'] & \\
\\
id \circ s & \to_{IdL} & s & \\
((t/x)\cdot s)\circ s' & \to_{Map} & (t[s']/x)\cdot(s\circ s') & \\
(s_1\circ s_2)\circ s_3 & \to_{Ass} & s_1\circ(s_2\circ s_3) &
\end{array}
$$

8.2 Towards strong normalization

We define the *Ateb* function as follows:

$$
\begin{array}{lcl}
Ateb(x) & = & x \\
Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\
Ateb(\lambda x.t) & = & \lambda x.Ateb(t) \\
Ateb(t[id]) & = & Ateb(t) \\
Ateb(t[s\circ s']) & = & Ateb(t[s][s']) \\
Ateb(t[(t'/x)\cdot s]) & = & Ateb((\lambda x.t)[s])\ Ateb(t')
\end{array}
$$

It is obvious that for any t, $Ateb(t)$ does not contain substitutions. We must check that the term we obtain is typeable.

Lemma 8.1

$$\Gamma \vdash t : A \;\Rightarrow\; \Gamma \vdash Ateb(t) : A$$

Proof: By induction on t.

- $t = x$ and

$$\frac{}{x : A, \Delta \vdash x : A}$$

  We have $Ateb(t) = x$ and the same typing derivation.

- $t = (u\ v)$ and

$$\frac{\Gamma \vdash u : B \to A \quad \Gamma \vdash v : B}{\Gamma \vdash (u\ v) : A}$$

  By induction hypothesis, we have $\Gamma \vdash Ateb(u) : B \to A$ and $\Gamma \vdash Ateb(v) : B$. We can type $Ateb(t) = Ateb(u)\ Ateb(v)$ as follows

$$\frac{\Gamma \vdash Ateb(u) : B \to A \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash (Ateb(u)\ Ateb(v)) : A}$$

- $t = \lambda x.u$ and

$$\frac{x : B, \Gamma \vdash u : A}{\Gamma \vdash \lambda x.u : B \to A}$$

By induction hypothesis, we have $\Gamma, x : B \vdash Ateb(u) : A$. We can type $Ateb(t) = \lambda x.Ateb(u)$ as follows

$$\frac{x : B, \Gamma \vdash Ateb(u) : A}{\Gamma \vdash \lambda x.Ateb(u) : B \to A}$$

- $t = u[id]$ and

$$\frac{\Gamma \vdash id \triangleright \Gamma \quad \Gamma \vdash u : A}{\Gamma \vdash u[id] : A}$$

We directly conclude by induction hypothesis.

- $t = u[(v/x) \cdot s]$ and

$$\frac{\dfrac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma \vdash v : B}{\Gamma \vdash (v/x) \cdot s \triangleright x : B, \Gamma'} \quad x : B, \Gamma' \vdash u : A}{\Gamma \vdash u[(v/x) \cdot s] : A}$$

By induction hypothesis, we have $\Gamma \vdash Ateb((\lambda x.u)[s]) : B \to A$ and $\Gamma \vdash Ateb(v) : B$. We conclude with the following typing derivation

$$\frac{\Gamma \vdash Ateb((\lambda x.u)[s]) : B \to A \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash Ateb((\lambda x.u)[s])\ Ateb(v) : A}$$

- $t = u[s \circ s']$, we conclude directly by induction hypothesis.

■

8.3 Definition of the relation $\lessdot$

We proceed as in the previous section, but more easily since there is no $\uparrow$. We use the same notion of redexability (see 7.10) and the relation $\preccurlyeq$ is define similarly (without the $\uparrow$). We define the relation $\lessdot$ as follows.

Definition 8.2 For all t and u, $u \lessdot t \iff u \preccurlyeq t$ and $\sigma(t) = \sigma(u)$.

We will use Lemma 7.13 and we need a new formulation of the Lemma 7.16.

Lemma 8.3 For all t, u, s and s' in σ-normal form, if $\sigma(t[(y/x) \cdot s]) = \sigma(t'[(y/x) \cdot s'])$ then $\sigma(t[(u/x) \cdot s]) = \sigma(t'[(u/x) \cdot s'])$.

Proof: Easy induction. ■

Here follows our initialization Lemma.

Lemma 8.4 (Initialisation) For all t, there exists u such that $Ateb(t) \rightarrow^*_B u$ and $u \lessdot t$.

Proof: By induction on t.

- If $t = x$, then $Ateb(t) = x$ and we conclude with $u = x$.
- If $t = (t_1\ t_2)$, then $Ateb(t) = (Ateb(t_1)\ Ateb(t_2))$. By induction hypothesis, there exists u_1 and u_2 such that $Ateb(t_1) \rightarrow^*_B u_1$ and $u_1 \lessdot t_1$ and $Ateb(t_2) \rightarrow^*_B u_2$ and $u_2 \lessdot t_2$. We conclude with $u = (u_1\ u_2)$.
- If $t = \lambda x.t'$, then $Ateb(t) = \lambda x.Ateb(t')$. By induction hypothesis, there exists u' such that $Ateb(t') \rightarrow^*_B u'$. We conclude with $u = \lambda x.u'$.
- If $t = t'[id]$, then $Ateb(t) = Ateb(t')$. By induction hypothesis, there exists u' such that $Ateb(t') \rightarrow^*_B u'$. We take $u = u'$ and we conclude with the help of Lemma 7.13.
- If $t = t'[s \circ s']$, then $Ateb(t) = Ateb(t'[s][s'])$. By induction hypothesis, there exists u' such that $Ateb(t'[s][s']) \rightarrow^*_B u'$. We conclude with $u = u'$.
- If $t = t_1[(t_2/x) \cdot s]$, then $Ateb(t) = Ateb((\lambda x.t_1)[s])\ Ateb(t_2)$. By induction hypothesis, there exists u_1 and u_2 such that $Ateb((\lambda x.t_1)[s]) \rightarrow^*_B u_1$ and $u_1 \lessdot (\lambda x.t_1)[s]$ and $Ateb(t_2) \rightarrow^*_B u_2$ and $u_2 \lessdot t_2$. There are two cases with respect to the form of u_1.
  - If $u_1 = \lambda x.v_1$ (and so $\neg PR(s)$), then we take $u = v_1[(u_2/x) \cdot id]$. We need to check that $u \lessdot t$ and the difficulty resides in the proof of $\sigma(u) = \sigma(t)$. By hypothesis, we have $\sigma(\lambda x.v_1) = \sigma((\lambda x.t_1)[s])$. It is obvious that $\sigma(\lambda x.v_1) = \sigma((\lambda x.v_1)[id]) = \lambda y.(\sigma(\sigma(v_1)[(y/x) \cdot id]))$. On the other hand, we have $\sigma((\lambda x.t_1)[s]) = \lambda y.(\sigma(\sigma(t_1)[(y/x) \cdot$

$\sigma(s)])$), which gives us $\sigma(\sigma(v_1)[(y/x)\cdot id]) = \sigma(\sigma(t_1)[(y/x)\cdot\sigma(s)])$. We can apply Lemma 8.3 with $\sigma(u_2)$ that is equal to $\sigma(t_2)$ by hypothesis, that gives us $\sigma(\sigma(v_1)[(\sigma(u_2)/x)\cdot id]) = \sigma(\sigma(t_1)[(\sigma(t_2)/x)\cdot s])$ and we can conclude.

– If $u_1 = (\lambda x.v_1)[s_1]$, then we take $u = v_1[(u_2/x)\cdot s_1]$ and we conclude similarly with the help of Lemma 8.3.

■

8.3.1 Simulation

The simulation will be similar to that defined for $\lambda\sigma$.

Lemma 8.5 (Simulation) For all t reducing with the rule B to t', for all $u \lessdot t$, there exists u' such that u reduces in one step to u' and $u' \lessdot t'$. For all the other rules, for all t reducing to t', for all $u \lessdot t$, there exists u' such that u reduces in zero or some steps to u' and $u' \lessdot t'$.

Proof: For all the rules except B (namely σ), the proof is simple. $u \lessdot t$ gives us $u \preccurlyeq t$ and $\sigma(u) = \sigma(t)$, and, on the other hand, $t \rightarrow_\sigma t'$ implies $\sigma(t) = \sigma(t')$. There are two cases with respect to the fact that the redex appears in u. If not, we take $u' = u$ and we conclude directly. Else, we reduce it with the same rule and we conclude with $\sigma(u') = \sigma(u) = \sigma(t')$.

For the rule B, it's more complicated. The hypothesis is he same, but we are sure that the redex appears in u, that was the point of defining the relation $\preccurlyeq$ with the help of the predicate PR. We then have $u \rightarrow_B u'$ and we want to prove $u' \lessdot t'$. Even if it is obvious that $u' \preccurlyeq t'$ comes directly from $u \preccurlyeq t$, it is not the case for the equality of the σ-normal forms. We want $\sigma(u') = \sigma(t')$ with the hypothesis $\sigma(u) = \sigma(t)$. We take $t = C[(\lambda x.v)\ w]$, which gives us $t' = C[v[(w/x)\cdot id]]$ and $u = C'[(\lambda x.v')\ w']$. Two cases are possible:

- the redex $(\lambda x.v)\ w$ does not appear in $\sigma(t)$. That means that the calculus of $\sigma(t)$ can be split as follows:

  $$C[(\lambda x.v)\,w] \rightarrow^*_\sigma C_1[y[((C_2[(\lambda x.v)\,w])/x)\cdot s]] \rightarrow_{VarCons2} C_1[y[s]] \rightarrow^*_\sigma \sigma(t)$$

  Since $\sigma(t) = \sigma(u)$, it occurs similarly for u. As for the redex, the reduct will be erased from t' and from u' and we get $\sigma(u') = \sigma(t')$.

- the redex $(\lambda x.v)\ w$ does appear in $\sigma(t)$. We will write, for all t, $\underline{t}$ for $\sigma(t)$, in order to clarify the presentation of the calculi. We have the following equalities:

$$\begin{array}{rcl}\underline{\text{t}} & = & \sigma(C[(\lambda x.v)\ w]) \\ & = & C_1[\sigma(((\lambda x.v)\ w)[\underline{\text{s}}])] \\ & = & C_1[(\lambda y.\sigma(\underline{\text{v}}[(y/x)\cdot \underline{\text{s}}]))\ \sigma(\underline{\text{w}}[\underline{\text{s}}])]\end{array}$$

And, similarly, $\underline{\text{u}} = C_1'[(\lambda y.\sigma(\underline{\text{v'}}[(y/x)\cdot\underline{\text{s'}}]))\ \sigma(\underline{\text{w'}}[\underline{\text{s'}}])]$. From $\underline{\text{t}} = \underline{\text{u}}$ we deduce $C_1 = C_1'$, $\sigma(\underline{\text{v}}[(y/x)\cdot\underline{\text{s}}]) = \sigma(\underline{\text{v'}}[(y/x)\cdot\underline{\text{s'}}])$ and $\sigma(\underline{\text{w}}[\underline{\text{s}}]) = \sigma(\underline{\text{w'}}[\underline{\text{s'}}])$. We now look at $\underline{\text{t'}}$ and $\underline{\text{u'}}$:

$$\begin{array}{rl}\underline{\text{t'}} = & \sigma(C[v[w\cdot id]]) \\ = & C_1[\sigma(\underline{\text{v}}[\underline{\text{w}}\cdot id][\underline{\text{s}}])] \\ =_{Clos} & C_1[\sigma(\underline{\text{v}}[(\underline{\text{w}}\cdot id)\circ\underline{\text{s}}])] \\ =_{Map} & C_1[\sigma(\underline{\text{v}}[\underline{\text{w}}[\underline{\text{s}}]\cdot\underline{\text{s}}])] \\ = & C_1[\sigma(\underline{\text{v}}[\sigma(\underline{\text{w}}[\underline{\text{s}}])\cdot\underline{\text{s}}])]\end{array}$$

And similarly, $\underline{\text{u'}} = C_1'[\sigma(\underline{\text{v'}}[\sigma(\underline{\text{w'}}[\underline{\text{s'}}])\cdot\underline{\text{s'}}])]$. From the preceding equalities, we deduce $\underline{\text{u'}} = C_1'[\sigma(\underline{\text{v'}}[\sigma(\underline{\text{w}}[\underline{\text{s}}])\cdot\underline{\text{s'}}])]$, and we can conclude with the help of Lemma 8.3.

■

Lemma 8.6 For all terms t, if $u \lessdot t$ and $u \in \Lambda^X_{SN}$, then $t \in \Lambda^X_{SN}$.

Proof: By the simulation Lemma 8.5, and since the σ-calculus is terminating [1], if we have an infinite derivation of t, then we can also build one in u, and that gives us a contradiction. ■

Since the $Ateb(t)$ function returns a term t' that reduces to $u \lessdot t$ (by Lemma 8.4), we know this technique can be applied to this calculus.

9 $\overline{\lambda}\mu\tilde{\mu}$-calculus

The $\overline{\lambda}\mu\tilde{\mu}$-calculus is a symmetric non-deterministic calculus that comes from classical logic. Its terms represent proof in classical sequent calculus. We can add to it explicit substitutions " la" λ**x**.

9.1 Definition

We have four syntactic categories: terms, contexts, commands and substitutions ; respectively denoted v, e, c and τ. We give to variable sets: Var is the set of term variables (denoted x, y, z etc.); $Var^\perp$ is the set of context variables (denoted α, β, γ etc.). We will denote by $*$ a variable for which the set to which it belong does not care, and by t an undetermined syntactic object among v, e and c.

The syntax of the $\overline{\lambda}\mu\tilde{\mu}$-calculus is given by the following grammar:

$$\begin{array}{l} c \ ::= \ \langle v|e\rangle \mid c\tau \\ v \ ::= \ x \mid \lambda x.v \mid e\cdot v \mid \mu\alpha.c \mid v\tau \\ e \ ::= \ \alpha \mid \alpha\lambda.e \mid v\cdot e \mid \tilde{\mu}x.c \mid e\tau \\ \tau ::= [x \leftarrow v] \mid [\alpha \leftarrow e] \end{array}$$

The source $Dom(\tau)$ of τ is x if $\tau = [x \leftarrow v]$ and α if $\tau = [\alpha \leftarrow e]$. The substituend $S(\tau)$ is v and e respectively.

The reduction rules are given below. Remark that the rules (μ) and $(\tilde{\mu})$ gives a critical pair:

$$\begin{array}{llcll}
(\beta) & \langle \lambda x.v|v'\cdot e\rangle & \rightarrow & \langle v'|\tilde{\mu}x.\langle v|e\rangle\rangle & \\
(\tilde{\beta}) & \langle e'\cdot v|\alpha\lambda.e\rangle & \rightarrow & \langle \mu\alpha.\langle v|e\rangle|e'\rangle & \\
(\mu) & \langle \mu\alpha.c|e\rangle & \rightarrow & c[\alpha \leftarrow e] & \\
(\tilde{\mu}) & \langle v|\tilde{\mu}x.c\rangle & \rightarrow & c[x \leftarrow v] & \\
 & & & & \\
(c\tau) & \langle v|e\rangle\tau & \rightarrow & \langle v\tau|e\tau\rangle & \\
(x\tau 1) & x[x \leftarrow v] & \rightarrow & v & \\
(x\tau 2) & x\tau & \rightarrow & x & \text{If } x \notin Dom(\tau) \\
(\alpha\tau 1) & \alpha[\alpha \leftarrow e] & \rightarrow & e & \\
(\alpha\tau 2) & \alpha\tau & \rightarrow & \alpha & \text{If } \alpha \notin Dom(\tau) \\
(\cdot\tau) & (v\cdot e)\tau & \rightarrow & (v\tau)\cdot(e\tau) & \\
(\tilde{\cdot}\tau) & (e\cdot v)\tau & \rightarrow & (e\tau)\cdot(v\tau) & \\
(\lambda\tau) & (\lambda x.v)\tau & \rightarrow & \lambda x.(v\tau) & \\
(\tilde{\lambda}\tau) & (\alpha\lambda.e)\tau & \rightarrow & \alpha\lambda.(e\tau) & \\
(\mu\tau) & (\mu\alpha.c)\tau & \rightarrow & \mu\alpha.(c\tau) & \\
(\tilde{\mu}\tau) & (\tilde{\mu}x.c)\tau & \rightarrow & \tilde{\mu}x.(c\tau) &
\end{array}$$

For the rules $(\mu\tau)$ and $(\tilde{\lambda}\tau)$ (resp. $(\tilde{\mu}\tau)$ and $(\lambda\tau)$) we might perform α-conversion on the bound variable α (resp. x) if necessary. We add two simplification rules:

$$\begin{array}{llcll}
(sv) & \mu\alpha.\langle v|\alpha\rangle & \rightarrow & v & \text{Si } \alpha \notin v \\
(se) & \tilde{\mu}x.\langle x|e\rangle & \rightarrow & e & \text{Si } x \notin e
\end{array}$$

Here follows the typing rules:

$$\frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : A \vdash \Delta}{\langle v|e\rangle : (\Gamma \vdash \Delta)}$$

$$\frac{}{\Gamma|\alpha : A \vdash \Delta, \alpha : A} \quad \frac{}{\Gamma, x : A \vdash \Delta|x : A}$$

$$\frac{\Gamma|e : B \vdash \alpha : A, \Delta}{\Gamma|\alpha\lambda.e : A - B \vdash \Delta} \quad \frac{\Gamma, x : A \vdash v : B|\Delta}{\Gamma \vdash \lambda x.v : A \to B|\Delta}$$

$$\frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : B \vdash \Delta}{\Gamma|v \cdot e : A \to B \vdash \Delta} \quad \frac{\Gamma \vdash v : B|\Delta \quad \Gamma|e : A \vdash \Delta}{\Gamma \vdash e \cdot v : A - B|\Delta}$$

$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma|\tilde{\mu}x.c : A \vdash \Delta} \quad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A|\Delta}$$

$$\frac{\Gamma \vdash v : A|\Delta}{[x \leftarrow v] : (\Gamma, x : A \vdash \Delta) \Rightarrow (\Gamma \vdash \Delta)} \quad \frac{\Gamma|e : A \vdash \Delta}{[\alpha \leftarrow e] : (\Gamma \vdash \alpha : A, \Delta) \Rightarrow (\Gamma \vdash \Delta)}$$

$$\frac{\Gamma|e : A \vdash \Delta \quad \tau : (\Gamma \vdash \Delta) \Rightarrow (\Gamma' \vdash \Delta')}{\Gamma'|e\tau : A \vdash \Delta'} \quad \frac{\Gamma \vdash v : A|\Delta \quad \tau : (\Gamma \vdash \Delta) \Rightarrow (\Gamma' \vdash \Delta')}{\Gamma' \vdash v\tau : A|\Delta'}$$

$$\frac{c : (\Gamma \vdash \Delta) \quad \tau : (\Gamma \vdash \Delta) \Rightarrow (\Gamma' \vdash \Delta')}{c\tau : (\Gamma' \vdash \Delta')}$$

9.2 Strong normalization

We define the *Ateb* function as follows:

$$\begin{array}{lcll}
Ateb(x) & = & x & \\
Ateb(\alpha) & = & \alpha & \\
Ateb(\langle v|e\rangle) & = & \langle Ateb(v)|Ateb(e)\rangle & \\
Ateb(\lambda x.v) & = & \lambda x.Ateb(v) & \\
Ateb(\alpha\lambda.e) & = & \alpha\lambda.Ateb(e) & \\
Ateb(\mu\alpha.c) & = & \mu\alpha.Ateb(c) & \\
Ateb(\tilde{\mu}x.c) & = & \tilde{\mu}x.Ateb(c) & \\
Ateb(e \cdot v) & = & Ateb(e) \cdot Ateb(v) & \\
Ateb(v \cdot e) & = & Ateb(v) \cdot Ateb(e) & \\
 & & & \\
Ateb(c[x \leftarrow v]) & = & \langle Ateb(v)|\tilde{\mu}x.Ateb(c)\rangle & \\
Ateb(c[\alpha \leftarrow e]) & = & \langle \mu\alpha.Ateb(c)|Ateb(e)\rangle & \\
Ateb(v[x \leftarrow v']) & = & \mu\alpha.\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle & \text{with } \alpha \text{ fresh} \\
Ateb(v[\alpha \leftarrow e]) & = & \mu\beta.\langle \mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle & \text{with } \beta \text{ fresh} \\
Ateb(e[x \leftarrow v]) & = & \tilde{\mu}y.\langle Ateb(v)|\tilde{\mu}x.\langle y|Ateb(e)\rangle\rangle & \text{with } y \text{ fresh} \\
Ateb(e[\alpha \leftarrow e']) & = & \tilde{\mu}x.\langle Ateb(e') \cdot x|\alpha\lambda.Ateb(e)\rangle & \text{with } x \text{ fresh}
\end{array}$$

It is obvious that for all t, $Ateb(t)$ does not contain substitutions. We must check firstly that the returned term is typeable, and secondly that it reduces to the original term.

Lemma 9.1

$$\Gamma \vdash t : A \;\; \Rightarrow \;\; \Gamma \vdash \mathit{Ateb}(t) : A$$

Proof: By induction of the typing derivation of t. The only interesting cases are those of substitutions.

- We type $c[x \leftarrow v]$

$$\dfrac{c : (\Gamma, x : A \vdash \Delta) \quad \dfrac{\Gamma \vdash v : A|\Delta}{[x \leftarrow v] : (\Gamma, x : A \vdash \Delta) \Rightarrow (\Gamma \vdash \Delta)}}{c[x \leftarrow v] : (\Gamma \vdash \Delta)}$$

  By induction hypothesis, we have $\mathit{Ateb}(c) : (\Gamma, x : A \vdash \Delta)$ and $\Gamma \vdash \mathit{Ateb}(v) : A|\Delta$. We can type $\mathit{Ateb}(c[x \leftarrow v]) = \langle \mathit{Ateb}(v)|\tilde{\mu}x.\mathit{Ateb}(c)\rangle$ as follows

$$\dfrac{\Gamma \vdash \mathit{Ateb}(v) : A|\Delta \quad \dfrac{\mathit{Ateb}(c) : (\Gamma, x : A \vdash \Delta)}{\Gamma|\tilde{\mu}x.\mathit{Ateb}(c) : A \vdash \Delta}}{\langle \mathit{Ateb}(v)|\tilde{\mu}x.\mathit{Ateb}(c)\rangle : (\Gamma \vdash \Delta)}$$

- The case $c[\alpha \leftarrow e]$ is similar to the previous one by symmetry.

- We type $v[x \leftarrow v']$

$$\dfrac{\Gamma, x : B \vdash v : A|\Delta \quad \dfrac{\Gamma \vdash v' : B|\Delta}{[x \leftarrow v'] : (\Gamma, x : B \vdash \Delta) \Rightarrow (\Gamma \vdash \Delta)}}{\Gamma \vdash v[x \leftarrow v'] : A|\Delta}$$

  By induction hypothesis, we have $\Gamma, x : B \vdash \mathit{Ateb}(v) : A|\Delta$ and $\Gamma \vdash \mathit{Ateb}(v') : B|\Delta$. We can type $\mathit{Ateb}(v[x \leftarrow v']) = \mu\alpha.\langle \lambda x.\mathit{Ateb}(v)|\mathit{Ateb}(v') \cdot \alpha\rangle$ as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, x : B \vdash \mathit{Ateb}(v) : A|\Delta}{\Gamma, x : B \vdash \mathit{Ateb}(v) : A|\Delta, \alpha : A}}{\Gamma \vdash \lambda x.\mathit{Ateb}(v) : B \rightarrow A|\Delta, \alpha : A} \quad \dfrac{\dfrac{\Gamma \vdash \mathit{Ateb}(v') : B|\Delta}{\Gamma \vdash \mathit{Ateb}(v') : B|\Delta, \alpha : A} \quad \Gamma|\alpha : A \vdash \Delta, \alpha : A}{\Gamma|\mathit{Ateb}(v') \cdot \alpha : B \rightarrow A \vdash \Delta, \alpha : A}}{\langle \lambda x.\mathit{Ateb}(v)|\mathit{Ateb}(v') \cdot \alpha\rangle : (\Gamma \vdash \Delta, \alpha : A)}}{\Gamma \vdash \mu\alpha.\langle \lambda x.\mathit{Ateb}(v)|\mathit{Ateb}(v') \cdot \alpha\rangle : A|\Delta}$$

- We type $v[\alpha \leftarrow e]$

$$\dfrac{\Gamma \vdash v : A|\Delta, \alpha : B \quad \dfrac{\Gamma \vdash e : B|\Delta}{[\alpha \leftarrow e] : (\Gamma \vdash \Delta, \alpha : B) \Rightarrow (\Gamma \vdash \Delta)}}{\Gamma \vdash v[\alpha \leftarrow e] : A|\Delta}$$

By induction hypothesis, we have $\Gamma \vdash Ateb(v) : A|\Delta, \alpha : B$ and $\Gamma \vdash Ateb(e) : B|\Delta$. We can type $Ateb(v[\alpha \leftarrow e]) = \mu\beta.\langle \mu\alpha.\langle Ateb(v)|\beta\rangle | Ateb(e)\rangle$ as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash Ateb(v) : A|\Delta, \alpha : B}{\Gamma \vdash Ateb(v) : A|\Delta, \beta : A, \alpha : B} \quad \Gamma|\beta : A \vdash \Delta, \beta : A, \alpha : B}{\langle Ateb(v)|\beta\rangle : (\Gamma \vdash \Delta, \beta : A, \alpha : B)}}{\Gamma \vdash \mu\alpha.\langle Ateb(v)|\beta\rangle : B|\Delta, \beta : A} \qquad \dfrac{\Gamma \vdash Ateb(e) : B|\Delta}{\Gamma \vdash Ateb(e) : B|\Delta, \beta : A}}{\langle \mu\alpha.\langle Ateb(v)|\beta\rangle | Ateb(e)\rangle : (\Gamma \vdash \Delta, \beta : A)}}{\Gamma \vdash \mu\beta.\langle \mu\alpha.\langle Ateb(v)|\beta\rangle | Ateb(e)\rangle : A|\Delta}$$

- The cases for $e[* \leftarrow t]$ are similar to the previous ones by symmetry.

$\blacksquare$

Lemma 9.2

$$Ateb(t) \to^* t$$

Proof: By induction on t. The only interesting cases are those of substitutions.

- We have $Ateb(c[x \leftarrow v]) = \langle Ateb(v)|\tilde{\mu}x.Ateb(c)\rangle$ and

  $$\langle Ateb(v)|\tilde{\mu}x.Ateb(c)\rangle \to_{\mu} Ateb(c)[x \leftarrow Ateb(v)]$$

  We conclude by induction hypothesis.

- The case $c[\alpha \leftarrow e]$ is similar to the previous one by symmetry.

- We have $Ateb(v[x \leftarrow v']) = \mu\alpha.\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle$ and

  $$\begin{array}{c} \mu\alpha.\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle \\ \downarrow \beta \\ \mu\alpha.\langle Ateb(v')|\tilde{\mu}x.\langle Ateb(v)|\alpha\rangle\rangle \\ \downarrow \tilde{\mu} \\ \mu\alpha.(\langle Ateb(v)|\alpha\rangle[x \leftarrow Ateb(v')]) \\ \downarrow c\tau \\ \mu\alpha.\langle Ateb(v)[x \leftarrow Ateb(v')]|\alpha[x \leftarrow Ateb(v')]\rangle \\ \downarrow \alpha\tau 2 \\ \mu\alpha.\langle Ateb(v)[x \leftarrow Ateb(v')]|\alpha\rangle \\ \downarrow sv \\ Ateb(v)[x \leftarrow Ateb(v')] \end{array}$$

  We conclude by induction hypothesis.

- We have $Ateb(v[\alpha \leftarrow e]) = \mu\beta.\langle\mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle$ and

$$\begin{array}{c}\mu\beta.\langle\mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle \\ \downarrow \mu \\ \mu\beta.(\langle Ateb(v)|\beta\rangle[\alpha \leftarrow Ateb(e)]) \\ \downarrow c\tau \\ \mu\beta.\langle Ateb(v)[\alpha \leftarrow Ateb(e)]|\beta[\alpha \leftarrow Ateb(e)]\rangle \\ \downarrow \alpha\tau 2 \\ \mu\beta.\langle Ateb(v)[\alpha \leftarrow Ateb(e)]|\beta\rangle \\ \downarrow sv \\ Ateb(v)[\alpha \leftarrow Ateb(e)]\end{array}$$

  We conclude by induction hypothesis.

- The cases for $e[* \leftarrow t]$ are similar to the previous ones by symmetry.

■

10 Conclusion

The technique formalized here gives a new tool to prove strong normalization of calculi with explicit substitutions. As we have seen, the principle of the proof technique is simple, and the difficulties arise in the definition of the reverse rewriting rule that must satisfy precise criteria.

We applied this technique to several calculi, yielding the following results:

- λx: there is here no novelty since it is this case that originally inspired the technique.
- λv: we gives here the first strong normalization proof for this calculus.
- $\lambda\sigma$: this calculus does not enjoy PSN, but we showed that no further objection relies to prove strong normalization.
- $\lambda\sigma_n$: as above.
- λ_{ws}: the technique seems to fail due to the presence of labels. Further investigations would be necessary to find how this can be fixed..
- λ_{wsn}: the technique can be used, even if this calculus has currently no proof of PSN.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1991.

[2] H. P. Barendregt. *The Lambda Calculus : its Syntax and Semantics.* 1981.

[3] Z.-E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 1996.

[4] R. Bloo. *Preservation of Termination for Explicit Substitution.* PhD thesis, Eindhoven University, 1997.

[5] R. Bloo and H. Geuvers. Explicit substitution: on the edge of strong normalisation. *Theoretical Computer Science*, 211:375–395, 1999.

[6] R. Bloo and K.H. Rose. Preservation of strong normalization in named lambda calculi with explicit substitution and garbage collection. *Computer Science in the Netherlands (CSN)*, 1995.

[7] A. Church. *The Calculi of Lambda Conversion.* Princeton University Press, 1941.

[8] R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. *Mathematical Structures in Computer Science*, 13(3):409–450, 2003.

[9] P.-L. Curien and A. Ríos. Un résultat de complétude pour les substitutions explicites. *Comptes rendus de l'académie des sciences de Paris*, t. 312, Série I:471–476, 1991.

[10] R. David and B. Guillaume. Strong normalisation of the typed λ_{ws}-calculus. In *Proceedings of CSL'03*, volume 2803 of *LNCS*. Springer, 2003.

[11] J.-L. Krivine. *Lambda-calcul, types et modles.* Masson, 1990.

[12] P. Lescanne. From lambda-sigma to lambda-upsilon: a journey through calculi of explicit substitutions. In *Proceedings of the 21st ACM Symposium on Principles of Programming Languages (POPL)*, pages 60–69, 1994.

[13] E. Polonovski. *Substitutions explicites, logique et normalisation.* Thèse de doctorat, Universit Paris VII, 2004.